ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

Αμυντικές Τεχνικές Παραπλάνησης σε Software Defined Networks (SDN)

Ηλίας Μπελαλής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Ιωάννης Αναγνωστόπουλος

Λαμία, 2020

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

Defensive Deception Techniques on Software Defined Networks (SDN)

Ilias Belalis

Master thesis

Ioannis Anagnostopoulos

Lamia

2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ

«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

Αμυντικές Τεχνικές Παραπλάνησης σε Software Defined Networks (SDN)

Ηλίας Μπελαλής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων

Ιωάννης Αναγνωστόπουλος

Λαμία, 2020

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Αμυντικές Τεχνικές Παραπλάνησης σε Software Defined Networks (SDN)» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ

Ημερομηνία

Υπογραφή

Αμυντικές Τεχνικές Παραπλάνησης σε Software Defined Networks (SDN)

Ηλίας Μπελαλής

Τριμελής Επιτροπή:

Πλαγιανάκος Βασίλειος,

Αναγνωστόπουλος Ιωάννης,

Κακαρούντας Αθανάσιος

Επιστημονικός Σύμβουλος:

Σπαθούλας Γεώργιος

Περιεχόμενα

Πίνακας Σχημάτων

Πίνακες

Εισαγωγή

Σήμερα, η πολυπλοκότητα και η λειτουργικότητα των δικτύων υπολογιστών αυξάνονται σημαντικά, αφού όλες οι συσκευές είναι συνδεδεμένες και προσβάσιμες από οπουδήποτε [3]. Ταυτόχρονα, η υλοποίηση αυτών των δικτύων είναι ζωτικής σημασίας για πολλούς τομείς, δεδομένου ότι χειρίζονται ροές δεδομένων προκειμένου να επιτύχουν διαφορετικές λειτουργίες. Δυστυχώς, υπάρχει πληθώρα επιθέσεων στον κυβερνοχώρο, όπως για παράδειγμα είναι οι επιθέσεις DDoS οι οποίες στοχεύουν τέτοια δίκτυα προκειμένου να εκτελέσουν κακόβουλες ενέργειες [4].

Αν και υπάρχουν μηχανισμοί που προσπαθούν να μετριάσουν τέτοιου είδους επιθέσεις [5], [6], κανένας μηχανισμός δεν καταφέρνει να διαχειριστεί και να προστατεύσει αποτελεσματικά δίκτυα μεγάλης κλίμακας. Επιπλέον, οι υπάρχουσες παραδοσιακές τεχνικές απαιτούν υψηλό κόστος για να υλοποιηθούν σε μεγάλα και σύνθετα δίκτυα.

Τα SDN είναι μια καινοτόμος τεχνολογία που στοχεύει να καταστήσει πιο ευέλικτη τη διαχείριση και τον προγραμματισμό των δικτύων και ειδικότερα των δικτύων μεγάλης κλίμακας. Συγκεκριμένα, μέχρι τώρα, στα παραδοσιακά δίκτυα το επίπεδο ελέγχου και το επίπεδο δεδομένων ήταν ομαδοποιημένα. Η τεχνολογία SDN διαχωρίζει το επίπεδο ελέγχου από το επίπεδο δεδομένων και δίνει τη δυνατότητα προγραμματισμού του δικτύου. Οι δρομολογητές και τα switches μπορούν να προγραμματιστούν μέσω του επιπέδου ελέγχου και επομένως η διαχείριση και η ανάπτυξη του δικτύου να γίνεται όλο και πιο εύκολη. Επιπλέον, πολλές από τις προκλήσεις των δικτύων υπολογιστών, όπως είναι η ανθεκτικότητα, η επεκτασιμότητα, οι επιδόσεις, η ασφάλεια και η αξιοπιστία, θα μπορούσαν να αντιμετωπιστούν με την τεχνολογία SDN.

Επιπλέον, για την προστασία των δικτύων από διάφορες επιθέσεις στον κυβερνοχώρο υπάρχουν διάφορες αμυντικές τεχνικές παραπλάνησης που θα μπορούσαν να χρησιμοποιηθούν σε συνδυασμό με την τεχνολογία SDN. Έχοντας μια τέτοια εφαρμογή αυξάνεται η ευρωστία και η επεκτασιμότητα των δικτύων μεγάλης κλίμακας. Συγκεκριμένα, οι υφιστάμενες αμυντικές τεχνικές παραπλάνησης, όπως τα Honeypots και η Moving Target Defense, δεν είναι τόσο αποτελεσματικές στα σύγχρονα, πολύπλοκα δίκτυα, δεδομένου ότι ο δυναμικός επαναπρογραμματισμός του δικτύου και η παρακολούθηση της ροής δεδομένων δεν είναι εφικτά. Ωστόσο τα SDN ξεπερνούν τέτοιους περιορισμούς και είναι κατάλληλα για την εφαρμογή αμυντικών τεχνικών παραπλάνησης στα σύγχρονα δίκτυα. [1]

Στη συνέχεια της εργασίας θα αναλυθεί η τεχνολογία SDN, θα παρουσιαστεί το πρωτόκολλο OpenFlow, θα εξεταστούν οι υπάρχουσες αμυντικές τεχνικές παραπλάνησης και οι εφαρμογές τους, θα παρουσιαστούν οι υπάρχουσες αμυντικές τεχνικές παραπλάνησης που χρησιμοποιούν την τεχνολογία SDN και στη συνέχεια θα συγκρίνουμε αυτές τις τεχνικές. Τέλος, θα παρουσιαστεί ένας μηχανισμός αμυντικής παραπλάνησης ο οποίος έχει αναπτυχθεί στον εξομοιωτή SDN δικτύων, Mininet. Στόχος του μηχανισμού αμυντικής παραπλάνησης είναι να αντιμετωπίζει δραστηριότητες

αναγνώρισης σε ένα δίκτυο, παρουσιάζοντας μια εικονική τοπολογία του δικτύου καθυστερώντας τον εντοπισμό ευάλωτων υπολογιστών στο δίκτυο από τους εισβολείς.

Εισαγωγή στα Software-Defined Networks

Ο κατανεμημένος έλεγχος και τα δικτυακά πρωτόκολλα που εκτελούνται στους δρομολογητές (routers) και στους μεταγωγείς (switches) επιτρέπουν την παροχή πληροφοριών, με τη μορφή ψηφιακών πακέτων. Παρά την ευρεία υιοθέτησή τους, τα παραδοσιακά IP δίκτυα είναι σύνθετα και αρκετά δύσκολα στη διαχείρισή τους.

Για να εφαρμόσουν τις επιθυμητές πολιτικές υψηλού επιπέδου, οι διαχειριστές των δικτύων πρέπει να διαμορφώσουν ξεχωριστά κάθε μεμονωμένη συσκευή δικτύου χρησιμοποιώντας εντολές χαμηλού επιπέδου οι οποίες συχνά διαφέρουν ανάλογα με τον κατασκευαστή ή και το μοντέλο της δικτυακής συσκευής. Επιπλέον, εκτός από την πολυπλοκότητα των ρυθμίσεων, τα δίκτυα πρέπει να ανταπεξέλθουν σε διάφορες βλάβες και να προσαρμοστούν στις αλλαγές φόρτου. Οι αυτοματοποιημένοι μηχανισμοί επαναπροσδιορισμού και απόκρισης είναι σχεδόν ανύπαρκτοι στα τρέχοντα IP δίκτυα. Συνεπώς, η επιβολή των απαιτούμενων πολιτικών σε ένα τέτοιο δυναμικό περιβάλλον είναι εξαιρετικά δύσκολη.

Κάθε δικτυακή συσκευή διαθέτει ενσωματωμένα (α) το επίπεδο ελέγχου (control plane), το οποίο είναι υπεύθυνο για τη διαχείριση της δικτυακής κίνησης και (β) το επίπεδο δεδομένων (data plane), το οποίο προωθεί τη δικτυακή κίνηση ανάλογα με τις αποφάσεις του επιπέδου ελέγχου. Έτσι, έχουμε μειωμένη ευελιξία και παρεμποδίζεται η καινοτομία και η εξέλιξη της δικτυακής υποδομής.

Η μετάβαση από το IPv4 στο IPv6, η οποία ξεκίνησε πριν από μια δεκαετία και εξακολουθεί να είναι σε μεγάλο βαθμό ατελής, αποτελεί ένα παράδειγμα του παραπάνω προβλήματος, ενώ στην πραγματικότητα το IPv6 αντιπροσώπευε απλώς μια ενημέρωση πρωτοκόλλου.

Η τεχνολογία Software-Defined Networking (SDN) προσφέρει την αλλαγή των περιορισμών των τωρινών δικτυακών υποδομών, δηλαδή τον διαχωρισμό της λογικής ελέγχου του δικτύου από τις συσκευές των δρομολογητών και των μεταγωγέων. Αυτός ο διαχωρισμός κάνει τις δικτυακές συσκευές, μόνο συσκευές προώθησης πακέτων, καθώς το επίπεδο ελέγχου υλοποιείται σε έναν κεντρικό ελεγκτή (Controller). Σαν αποτέλεσμα έχουμε την εύκολη υλοποίηση πολιτικών, ρυθμίσεων και εξέλιξης του δικτύου.

Ο παραπάνω διαχωρισμός πραγματοποιείται μέσω μιας προγραμματιζόμενης διεπαφή (interface) μεταξύ των μεταγωγέων και του SDN ελεγκτή. Ο ελεγκτής επιτρέπει τον έλεγχο της κατάστασης των στοιχείων που αποτελούν το επίπεδο δεδομένων μέσω μίας προγραμματιζόμενης διεπαφής εφαρμογής (API), όπως για παράδειγμα είναι το OpenFlow.

Αν και οι τεχνολογίες SDN και OpenFlow ξεκίνησαν πειραματικά από την ακαδημαϊκή κοινότητα, τα τελευταία χρόνια τράβηξαν την προσοχή της βιομηχανίας και οι περισσότεροι κατασκευαστές δικτυακών συσκευών περιλαμβάνουν την υποστήριξη του API OpenFlow στον εξοπλισμό τους. Η δυναμική του SDN ήταν αρκετά ισχυρή ώστε να κάνει την Google, το Facebook, τη Yahoo, τη Microsoft, τη Verizon και την Deutsche Telekom να χρηματοδοτήσουν το Open Networking Foundation (ONF) με κύριο στόχο την εξέλιξη, την προώθηση και την αποδοχή του SDN.

Ο όρος SDN επινοήθηκε αρχικά από το Standford University για αντιπροσωπεύσει τις ιδέες και την εργασία γύρω από το πρωτόκολλο OpenFlow. Όπως ορίστηκε αρχικά, το SDN αναφέρεται σε μια αρχιτεκτονική δικτύου όπου η προώθηση της κίνησης στο επίπεδο δεδομένων ελέγχεται από ένα απομακρυσμένο - αποσυνδεδεμένο επίπεδο ελέγχου. Επειδή η βιομηχανία των δικτύων έχει επανειλημμένα αποκλίνει από την αρχική έννοια των SDN, αναφέροντας οτιδήποτε αφορά λογισμικό ως SDN θα ορίζουμε το SDN ως μια αρχιτεκτονική δικτύου στην οποία:

- Το επίπεδο ελέγχου είναι διαχωρισμένο από το επίπεδο δεδομένων.
- Οι αποφάσεις για την προώθηση των πακέτων γίνονται με βάση τις ροές που έχουν εγκατασταθεί από τον ελεγκτή και όχι με βάση τον προορισμό του πακέτου.
- Η λογική του ελέγχου μετακινείται σε μια εξωτερική οντότητα που ονομάζεται ελεγκτής SDN (SDN Controller) ή Network Operating System (NOS).
- Το δίκτυο είναι προγραμματιζόμενο μέσω εφαρμογών που εκτελούνται στον ελεγκτή και αλληλεπιδρά με το επίπεδο δεδομένων. [1][2]

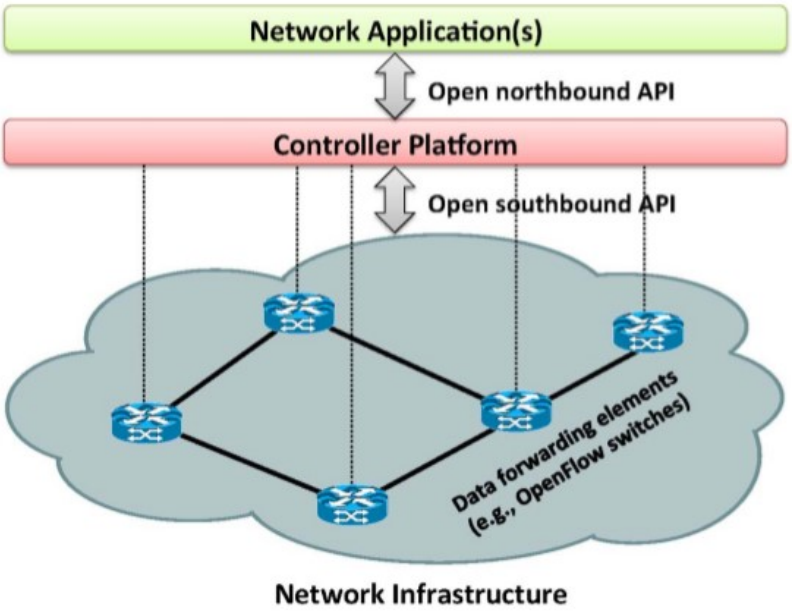


Figure 1: Απλοποιημένη όψη της αρχιτεκτονικής SDN (Reference: [2]).

Αρχιτεκτονική SDN

Η αρχιτεκτονική του SDN μπορεί να περιγραφεί ως μια σύνθεση των διαφορετικών στρωμάτων δικτύου, όπως φαίνεται στην παρακάτω εικόνα. Κάθε στρώμα έχει συγκεκριμένες λειτουργίες, όπως για παράδειγμα είναι τα Southbound API, NOSs, Northbound API και Network Applications. [2]

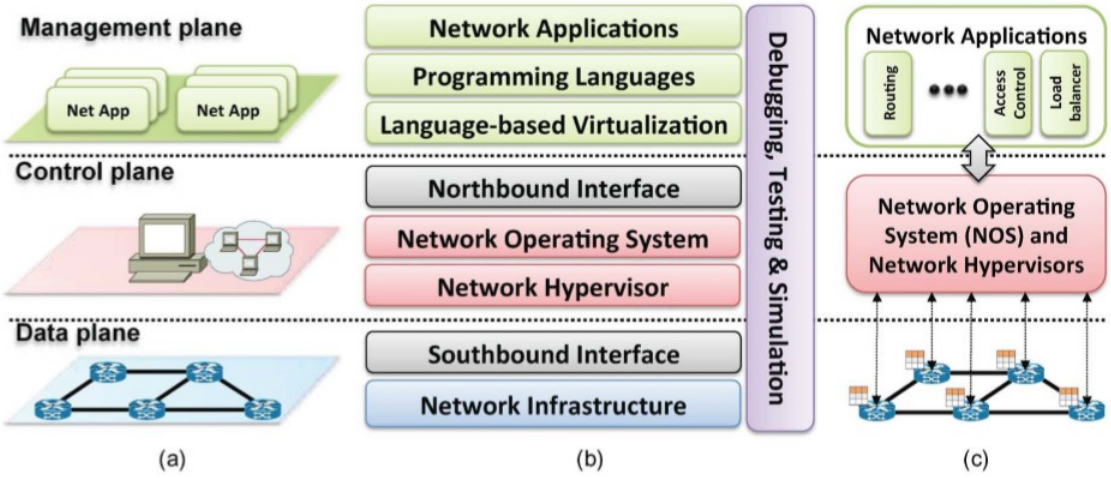


Figure 2: (a) Αρχιτεκτονική SDN σε επίπεδα, (b) σε στρώματα, (c) σχεδιασμό συστήματος (Reference: [2]).

Υποδομή δικτύου SDN (Network Infrastructure)

Η υποδομή στα SDN δίκτυα όπως και στα παραδοσιακά δίκτυα αποτελείται από δικτυακές συσκευές όπως είναι οι δρομολογητές και οι μεταγωγείς (switches) κλπ. Η σημαντικότερη διαφορά της υποδομής των SDN δικτύων από την υποδομή των παραδοσιακών δικτύων είναι ότι οι δικτυακές συσκευές είναι στοιχεία προώθησης της δικτυακής κίνησης χωρίς να έχουν ενσωματωμένο λογισμικό για τον έλεγχο της προώθησης των πακέτων. Θα μπορούσαμε να πούμε ότι ο «εγκέφαλος του δικτύου» έχει αφαιρεθεί από τις δικτυακές συσκευές και βρίσκεται σε ένα κεντρικό σύστημα ελέγχου το οποίο ονομάζεται Network Operating System (NOS).

Το σημαντικότερο είναι ότι τα SDN δίκτυα είναι σχεδιασμένα να λειτουργούν πάνω από ανοικτές και τυποποιημένες διεπαφές, όπως είναι το OpenFlow, το οποίο διασφαλίζει τη συμβατότητα επικοινωνίας και διαμόρφωσης (configuration) μεταξύ διαφορετικών συσκευών στο επίπεδο δεδομένων και στο επίπεδο ελέγχου. Με άλλα λόγια, αυτές οι διεπαφές επιτρέπουν στον ελεγκτή να προγραμματίζει δυναμικά ετερογενείς συσκευές προώθησης, κάτι που είναι δυσκολο στα παραδοσιακά δίκτυα λόγω της μεγάλης ποικιλίας κατασκευαστών, των κλειστών διεπαφών και της κατανεμημένης φύσης του επιπέδου ελέγχου.

Στην αρχιτεκτονική SDN υπάρχουν δύο κύρια στοιχεία, οι ελεγκτές και οι συσκευές προώθησης. Μια συσκευή επιπέδου δεδομένων ειδικεύεται στην προώθηση των πακέτων, ενώ ο ελεγκτής είναι λογισμικό το οποίο εκτελείται σε ένα απομακρυσμένο μηχάνημα. Μια συσκευή συμβατή με το πρωτόκολλο OpenFlow διατηρεί πίνακες ροής, όπου κάθε εγγραφή του πίνακα εξυπηρετεί τις παρακάτω τρείς λειτουργίες:

- Αντιστοίχηση των ροών με τους κανόνες,
- Περιέχει τις ενέργειες που θα εκτελεστούν σε περίπτωση αντιστοίχισης (match) των πακέτων,
- Περιέχει μετρητές οι οποίοι κρατούν στατιστικά στοιχεία των πακέτων.

Σε μια δικτυακή συσκευή συμβατή με το OpenFlow, μέσω μιας αλληλουχίας από πίνακες ροής δημιουργείται ένα μονοπάτι το οποίο καθορίζει τον τρόπο χειρισμού των πακέτων. Όταν φτάσει ένα νέο πακέτο στο switch, η διαδικασία αναζήτησης αρχίζει από τον πρώτο πίνακα ροής και τελειώνει είτε με μια αντιστοίχιση σε έναν από τους πίνακες ροής είτε με μια αστοχία (miss) όταν δεν υπάρχει κανόνας για το συγκεκριμένο πακέτο. Ένας κανόνας ροής μπορεί να αποτελείται από το συνδυασμό διαφορετικών πεδίων όπως απεικονίζεται στο παρακάτω σχήμα. Εάν δεν υπάρχει κανένας κανόνας ροής για κάποιο πακέτο τότε το πακέτο θα απορριφθεί. Ωστόσο, μπορούμε να εγκαταστήσουμε ένα κανόνα ώστε το switch να στείλει τα πακέτα για τα οποία δεν υπάρχει αντιστοίχιση στον ελεγκτή. Οι προτεραιότητες των κανόνων ακολουθούν την αλληλουχία του αριθμού των πινάκων ροής καθώς και της σειράς σε ένα πίνακα ροής. Οι δυνατές ενέργειες που μπορούν να εφαρμοστούν σε ένα πακέτο είναι:

- Να προωθηθεί το πακέτο σε κάποια πόρτα της δικτυακής συσκευής,
- Να ενθυλακωθεί το πακέτο και να προωθηθεί στον ελεγκτή,
- Να απορριφθεί (drop) το πακέτο,
- Να σταλεί σε έναν επόμενο πίνακα ροής ή σε καποιον από τους ειδικούς πίνακες όπως οι πίνακες meter που εισάγονται στην τελευταία έκδοση του πρωτοκόλλου OpenFlow. [2]

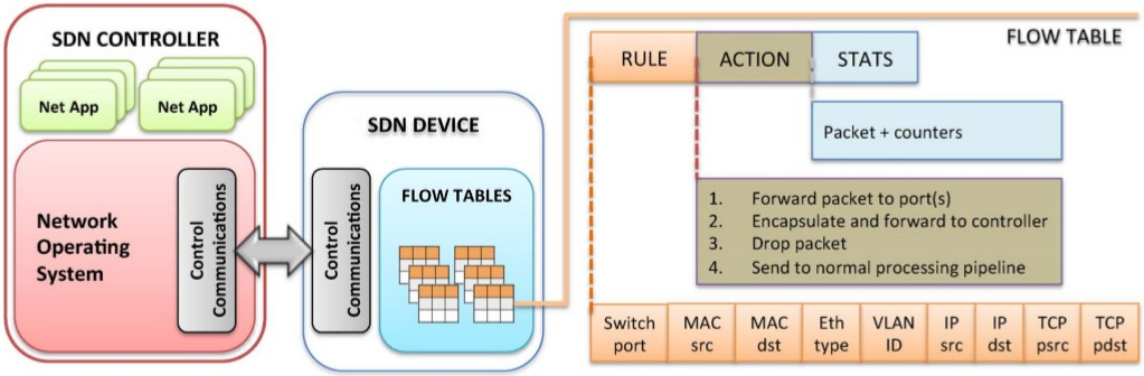


Figure 3: Συσκευές που υποστηρίζουν το πρωτόκολλο OpenFlow (Reference: [2]).

Southbound Interfaces

Το Southbound Interface ή Southbound API αποτελεί τη γέφυρα επικοινωνίας ανάμεσα στο επίπεδο έλεγχου και στο επίπεδο δεδομένων, δηλαδή τις συσκευές προώθησης. Αυτή τη στιγμή, το OpenFlow είναι η ευρέως αποδεκτή λύση και αναπτύσσεται σαν ανοικτό πρότυπο Southbound API για το SDN. Παρέχει κοινές προδιαγραφές για τις OpenFlow συσκευές προώθησης και για την επικοινωνία μεταξύ των συσκευών του επιπέδου δικτύου και του επιπέδου ελέγχου μέσω ενός καναλιού.

Σύμφωνα με το πρωτόκολλο OpenFlow ο ελεγκτής λαμβάνει τις παρακάτω πληροφορίες:

- Από τις συσκευές προώθησης αποστέλλονται στον ελεγκτή μηνύματα όταν μια σύνδεση ή μια πόρτα παρουσιάσει κάποια αλλαγή, δηλαδή μηνύματα βασισμένα σε γεγονότα (event-based messages).
- Από τις συσκευές προώθησης δημιουργούνται στατιστικά στοιχεία ροών τα οποία στη συνέχεια στέλνονται στον ελεγκτή.
- Από τις συσκευές προώθησης αποστέλλονται στον ελεγκτή μηνύματα, τύπου packet-in, όταν δεν γνωρίζουν πώς να χειριστούν μια νέα εισερχόμενη ροή ή επειδή υπάρχει καθορισμένη ενέργεια τύπου “send to controller” αντιστοιχισμένη στον πίνακα ροής. [2]

Network Operating Systems (NOS) / Controllers

Τα Λειτουργικά Συστήματα παρέχουν αρκετές αφαιρέσεις όπως είναι τα υψηλού επιπέδου APIs, για την πρόσβαση σε συσκευές χαμηλότερου επιπέδου, για παράδειγμα τον σκληρό δίσκο, την κάρτα δικτύου, την CPU, την μνήμη RAM, καθώς επίσης παρέχουν και τους απαραίτητους μηχανισμούς ασφάλειας. Οι παραπάνω λειτουργίες είναι οι βασικοί παράγοντες που επιτρέπουν την αύξηση της παραγωγικότητας, κάνοντας τη ζωή των προγραμματιστών ευκολότερη. Η ευρεία χρήση τους έχει συμβάλει στην ανάπτυξη πολλών εφαρμογών και στην εξέλιξη των γλωσσών προγραμματισμού. Σε αντίθεση, τα δίκτυα διαχειρίζονται και ρυθμίζονται από χαμηλού επιπέδου σύνολα οδηγιών προσαρμοσμένα στο υλικό από κλειστά λειτουργικά συστήματα όπως είναι για παράδειγμα το Cisco IOS και το Juniper Junos. Επιπλέον, η λογική των Λειτουργικών Συστημάτων να αφαιρούν χαρακτηριστικά συγκεκριμένων συσκευών και να παρέχουν, με διαφανή τρόπο, κοινές λειτουργίες είναι κάτι που απουσιάζει από τα δίκτυα.

Το SDN διευκολύνει τη διαχείριση των δικτύων μέσω του λογικά δομημένου κεντρικού ελέγχου που προσφέρεται από ένα NOS. Όπως στα Λειτουργικά Συστήματα των ηλεκτρονικών υπολογιστών, έτσι και ένα NOS θα πρέπει να παρέχει τις παραίτητες αφαιρέσεις, τις βασικές υπηρεσίες και κοινά APIs στους προγραμματιστές. Οι σημαντικότερες υπηρεσίες που παρέχονται

από ένα NOS είναι η παρακολούθηση της κατάστασης του δικτύου, πληροφορίες σχετικές με τη δικτυακή τοπολογία, η αναζήτηση συσκευών και η διανομή των δικτυακών ρυθμίσεων. Με το NOS, ο προγραμματιστής δεν χρειάζεται πλέον να νοιάζεται για λεπτομέρειες χαμηλού επιπέδου όπως η διανομή ρυθμίσεων μεταξύ των στοιχείων δρομολόγησης ώστε να καθορίσει τις πολιτικές του δικτύου. Τέτοια συστήματα μπορούν να ενισχύσουν την καινοτομία με ταχύτερο ρυθμό, μειώνοντας την εν γένη πολυπλοκότητα της δημιουργίας νέων πρωτοκόλλων δικτύου και δικτυακών εφαρμογών. [2]

Northbound Interfaces

Οι διεπαφές Northbound και Southbound αποτελούν τις δύο βασικές αφαιρέσεις του συστήματος SDN. Όπως είδαμε η διεπαφή Southbound έχει ήδη μια ευρέως αποδεκτή πρόταση, η οποία είναι το πρωτόκολλο OpenFlow. Η διεπαφή Northbound εξακολουθεί να είναι ένα ανοικτό ζήτημα αφού μέχρι αυτή τη στιγμή δεν έχει οριστεί κάποιο πρότυπο καθώς οι περιπτώσεις χρήσης εξακολουθούν να είναι σε επεξεργασία. [2]

Network Applications

Οι εφαρμογές των δικτύων μπορούμε να πούμε ότι είναι ο «εγκέφαλος του δικτύου» αφού εφαρμόζουν τη λογική ελέγχου η οποία θα μεταφραστεί σε κανόνες που θα εγκατασταθούν στο επίπεδο δεδομένων, καθορίζοντας την συμπεριφορά των συσκευών προώθησης. Ως παράδειγμα μπορούμε να δούμε μία απλή εφαρμογή δρομολόγησης. Η λογική της εφαρμογής αυτής είναι να καθορίσει τη διαδρομή μέσω της οποίας τα πακέτα θα ρέουν από το σημείο Α στο σημείο Β. Για να επιτευχθεί αυτός ο στόχος, μια εφαρμογή δρομολόγησης πρέπει με βάση την τοπολογία να αποφασίσει για τη διαδρομή που θα χρησιμοποιηθεί και να δώσει εντολή στον ελεγκτή να εγκαταστήσει τους αντίστοιχους κανόνες προώθησης σε όλες τις συσκευές προώθησης στην επιλεγμένη διαδρομή από το Α στο Β.

Το SDN μπορεί να αναπτυχθεί σε οποιοδήποτε περιβάλλον δικτύου, από οικιακά δίκτυα και δίκτυα επιχειρήσεων έως δίκτυα Data Centers και Internet Exchange Points. Μία τέτοια ποικιλία περιβαλλόντων οδήγησε σε ένα ευρύ φάσμα εφαρμογών δικτύου. Οι υπάρχουσες εφαρμογές δικτύου εκτελούν παραδοσιακές λειτουργίες, όπως δρομολόγηση, εξισορρόπηση φορτίου (Load Balancing), πολιτικές ασφαλείας, αλλά και διερεύνηση νέων προσεγγίσεων, όπως η μείωση κατανάλωσης ενέργειας. Άλλα παραδείγματα περιλαμβάνουν λειτουργίες fail-over, λειτουργίες αξιοπιστίας στο επίπεδο των δεδομένων, end-to-end QoS, Virtualization του δικτύου και διαχείριση

της κινητικότητας σε ασύρματα δίκτυα. Η ποικιλία των εφαρμογών δικτύου σε συνδυασμό με την ανάπτυξη πραγματικών περιπτώσεων χρήσης, αναμένονται να είναι από τις σημαντικότερες λόγους για την προώθηση και την ευρεία υιοθέτηση των SDN. Παρά τη μεγάλη ποικιλία των περιπτώσεων χρήσης, οι περισσότερες εφαρμογές SDN μπορούν να ομαδοποιηθούν στις παρακάτω κατηγορίες:

- Μηχανισμός δικτυακής κίνησης (Traffic Engineering)
- Κινητικότητα και Ασύρματη σύνδεση (Mobility and Wireless)
- Μετρήσεις και παρακολούθηση δικτύου (Measurement and Monitoring)
- Ασφάλεια και αξιοπιστία δικτύου (Security and Dependability)
- Δικτύωση Κέντρου Δεδομένων (Data Center Networking) [2]

Το πρωτόκολλο OpenFlow

Σε αυτό το κεφάλαιο θα περιγράψουμε τα χαρακτηριστικά του πρωτοκόλλου OpenFlow και θα εξετάσουμε τις διαφορετικές εκδόσεις του επισημαίνοντας τις υποστηριζόμενες λειτουργίες καθώς και τις αλλαγές μεταξύ των εκδόσεων.

Το πρωτόκολλο OpenFlow είναι το πρωτόκολλο της Southbound διεπαφής των SDN, το οποίο συνδέει το επίπεδο δεδομένων και το επίπεδο ελέγχου. Αρχικά, το OpenFlow προτάθηκε από το Πανεπιστήμιο του Stanford και τυποποιήθηκε από το ONF (Open Networking Foundation).

Η αρχιτεκτονική OpenFlow αποτελείται από τρεις βασικές έννοιες:

- Το δίκτυο το οποίο δημιουργείται από switches συμβατά με το OpenFlow τα οποία συνθέτουν το επίπεδο δεδομένων.
- Το επίπεδο ελέγχου το οποίο αποτελείται από έναν ή περισσότερους ελεγκτές OpenFlow.
- Ένα ασφαλές κανάλι ελέγχου το οποίο συνδέει τα switches με το επίπεδο ελέγχου. Το switch και ο ελεγκτής επικοινωνούν μέσω μιας σύνδεσης TLS η οποία δημιουργείται απο το switch κατά την εκκίνηση προς τον ελεγκτή στην TCP πόρτα 6633.

OpenFlow Switch

Ένας μεταγωγέας (switch) συμβατό με το πρωτόκολλο OpenFlow αποτελείται από έναν ή περισσότερους πίνακες ροής (flow table), όπου εκτελούνται διαδικασίες αντιστοίχισης και προώθησης στα πακέτα και από ένα ασφαλές κανάλι επικοινωνίας με έναν εξωτερικό ελεγκτή.

Ο ελεγκτής διαχειρίζεται το switch μέσα από το ασφαλές κανάλι χρησιμοποιώντας το πρωτόκολλο OpenFlow. Ο πίνακας ροής περιέχει ένα σύνολο εγγραφών που περιέχουν τις τιμές κεφαλίδων των πακέτων, τους μετρητές, και τις ενέργειες σε περίπτωση αντιστοίχισης των πακέτων. Όλα τα πακέτα επεξεργάζονται από το switch και συγκρίνονται με τον πίνακα ροής.

Εάν βρεθεί αντιστοίχιση ενός πακέτου με μια εγγραφή του πίνακα ροής, τότε εκτελούνται στο πακέτο οι ενέργειες που ισχύουν για την συγκεκριμένη εγγραφή. Για παράδειγμα μία ενέργεια είναι η προώθηση του πακέτου σε μία συγκεκριμένη πόρτα εξόδου του switch. Εάν δεν βρεθεί καμία εγγραφή που να ταιριάζει, τότε το πακέτο διαβιβάζεται στον ελεγκτή μέσα από το ασφαλές κανάλι.

Ο ελεγκτής είναι υπεύθυνος να χειρίζεται τα πακέτα τα οποία δεν έχουν έγκυρες εγγραφές στον πίνακα ροής, καθώς και να διαχειρίζεται τον πίνακα ροής του switch με την προσθήκη και την αφαίρεση εγγραφών.

OpenFlow 1.0

Το Open Flow 1.0 [48] κυκλοφόρησε τον Δεκέμβριο του 2009. Η έκδοση αυτή είναι η πιο διαδεδομένη του OpenFlow. Ένα switch συμβατό με το OpenFlow είναι μια δικτυακή συσκευή η οποία προωθεί τα πακέτα σύμφωνα με έναν πίνακα ροής. Ο συγκεκριμένος πίνακας περιέχει ένα σύνολο κανόνων, καθένας από τους οποίους αποτελείται από τα πεδία αντιστοίχισης, τους μετρητές και τις οδηγίες, όπως απεικονίζεται στο Σχήμα 4.

Τα «πεδία κεφαλίδας» σε μια εγγραφή του πίνακα ροής περιγράφουν τα πακέτα για τα οποία ισχύει αυτή η καταχώρηση. Το συγκεκριμένο πεδίο περιέχει τους κανόνες με τους οποίους συκρίνεται κάθε εισερχόμενο πακέτο στη δικτυακή συσκευή. Κάθε πεδίο μπορεί να περιέχει μια συγκεκριμένη τιμή (extract rule), ή την τιμή any (wildcard rule), που αντιστοιχεί με οποιαδήποτε τιμή του πακέτου. Για τους κανόνες που έχουν όλα τα πεδία καθορισμένα μπορούμε να έχουμε το πολύ έναν κανόνα για κάθε ενεργή ροή (active flow). Στους κανόνες wildcard πολλοί κανόνες μπορούν να αντιστοιχούν σε ένα πακέτο και έτσι θα πρέπει να οριστούν προτεραιότητες ώστε να ξεκαθαριστεί ποιος κανόνας θα εφαρμοστεί κάθε φορά.

Η λειτουργία των «μετρητών» είναι η συλλογή στατιστικών στοιχείων των ροών καθώς αποθηκεύουν τον αριθμό των ληφθέντων πακέτων, τα bytes, καθώς και τη διάρκεια της κάθε ροής. Οι μετρητές διατηρούνται και ανανεώνονται για κάθε πίνακα ροής, εγγραφή, θύρα και ουρά του switch. Οι «ενέργειες» καθορίζουν τον τρόπο χειρισμού των πακέτων της ροής. Οι ενέργειες μπορεί να είναι forward, drop, modify field, κλπ.

Header Fields	Counters	Actions

Figure 4: Εγγραφή του πίνακα ροής του πρωτοκόλλου Open Flow 1.0 (Reference: [47])

Όπως αναφέραμε και παραπάνω, ο ελεγκτής, είναι υπεύθυνος για τη δημιουργία και τον χειρισμό των πινάκων ροής των switches. Με την εισαγωγή, την τροποποίηση και την αφαίρεση των κανόνων ροής, ο ελεγκτής μπορεί να τροποποιήσει τη συμπεριφορά των switches όσον αφορά την προώθηση των πακέτων. Το OpenFlow ορίζει το πρωτόκολλο που επιτρέπει στον ελεγκτή να καθοδηγήσει τα switches.

Υπάρχουν τρεις τύποι επικοινωνίας στο πρωτόκολλο OpenFlow:

- controller-to-switch επικοινωνία,
- Ασύγχρονη (Asynchronous) επικοινωνία και
- Συμμετρική (symmetric)επικοινωνία.

Η επικοινωνία **controller-to-switch** είναι υπεύθυνη για την ανίχνευση χαρακτηριστικών, τη διαμόρφωση, τον προγραμματισμό του switch, καθώς και την ανάκτηση πληροφοριών από τα

switches. Τα μηνύματα αποστέλλονται από τον ελεγκτή στο switch και μπορεί να περιμένουν ή όχι απάντηση από το switch.

- Feauters: Κατά την εγκαθίδρυση της σύνδεσης ο ελεγκτής στέλνει μήνυμα στο switch και ζητάει τα χαρακτηριστικά που υποστηρίζει (feauters request). Στη συνέχεια, το switch απαντάει με τα χαρακτηριστικά του (feauters reply).
- Configuration: Ο ελεγκτής μπορεί να ζητήσει πληροφορίες σχετικά με τις υπάρχουσες ρυθμίσεις ενός switch ή να προβεί στη ρύθμιση των παραμέτρων του.
- Modify – State: Διαχείριση της κατάστασης του switch με την προσθήκη, τη διαγραφή και την τροποποίηση των εγγραφών του πίνακα ροής καθώς και τη ρύθμιση των θυρών του switch.
- Read –State: Συλλογή στατιστικών δεδομένων από τους πίνακες ροής και τις θύρες του switch.
- Barrier: Χρησιμοποιείται είτε για τη λήψη ειδοποιήσεων μετά την ολοκλήρωση κάποιων ενεργειών, είτε για να βεβαιωθεί πως οι εξαρτήσεις προηγούμενων μηνυμάτων έχουν ολοκληρωθεί πριν την επεξεργασία των επόμενων μηνημάτων.

Η **ασύγχρονη επικοινωνία** ξεκινάει από το OpenFlow switch χωρίς καμία παρακίνηση από τον ελεγκτή. Χρησιμοποιείται για την ενημέρωση του ελεγκτή σχετικά με τις αφίξεις νέων πακέτων, τις αλλαγές κατάστασης στο switch και τα διάφορα σφάλματα. Χωρίζεται στα ακόλουθα μηνύματα:

- Packet – in: Τα πακέτα που δεν αντιστοιχούν σε κάποια εγγραφή του πίνακα ροής προκαλούν την αποστολή ενός μηνύματος τύπου packet-in. Εάν το switch διαθέτει επαρκή διαθέσιμη μνήμη στον buffer μπορεί να κρατήσει προσωρινά το πακέτο και το packet-in μήνυμα περιλαμβάνει ένα τμήμα του πακέτου. Από προεπιλογή το συγκεκριμένο μήνυμα αποτελείται από τα πρώτα 128 bytes και μια αναγνωριστική τιμή στον buffer. Σε περίπτωση μη υποστήριξης προσωρινής αποθήκευσης ή μη επαρκούς μνήμης, το πακέτο στέλνεται ολόκληρο στον ελεγκτή.
- Flow – Removed: Κατά την προσθήκη μιας εγγραφής στον πίνακα ροής του switch από τον ελεγκτή, υπάρχουν δυο πεδία σχετικά με την αυτόματη αφαίρεση της συγκεκριμένης εγγραφής. Το πρώτο πεδίο είναι το soft timeout και αφαιρεί την εγγραφή μετά από κάποιο χρονικό διάστημα αδράνειας, δηλαδή καμίας αντιστοίχισης πακέτων με αυτή την εγγραφή. Το δεύτερο πεδίο είναι το hard timeout και προσδιορίζει μετά από ποιο χρονικό διάστημα θα αφαιρεθεί η εγγραφή χωρίς να χρησιμοποιηθεί κάποιο άλλο κριτήριο. Στην εγγραφή επίσης υπάρχει μια σημαία (flag) η οποία ορίζει εάν η αφαίρεση πρέπει να προκαλέσει την αποστολή μηνύματος Flow–Removed στον controller.

- Ports – status: Ένα switch στέλνει τέτοια μηνύματα κατά την αλλαγή της κατάστασης σε μια θύρα του. Για παράδειγμα αλλαγές από το spanning tree protocol.
- Error: Ενημέρωση του ελεγκτή για τυχόν σφάλματα που παρουσιάστηκαν στο swich.

Τέλος, αποστέλλονται **συμμετρικά μηνύματα** από τις δύο πλευρές, όπου το switch ή ο ελεγκτής είναι ελεύθεροι να εκκινήσουν την επικοινωνία χωρίς να το ζητήσει η άλλη πλευρά. Χωρίζονται στα ακόλουθα μηνύματα:

- Hello: Ανταλλάσσονται ανάμεσα στο switch και τον ελεγκτή κατά την εκκίνηση της σύνδεσης.
- Echo: Μηνύματα echo request στέλνονται είτε από το switch είτε από τον ελεγκτή και περιμένουν ένα echo reply σε απάντηση. Χρησιμοποιούνται σαν δείκτες για latency/bandwidth, καθώς επίσης και για έλεγχο της σύνδεσης.
- Vendor: Αποτελούν ένα τρόπο επέκτασης εντός του πρωτοκόλλου OpenFlow.

[47]

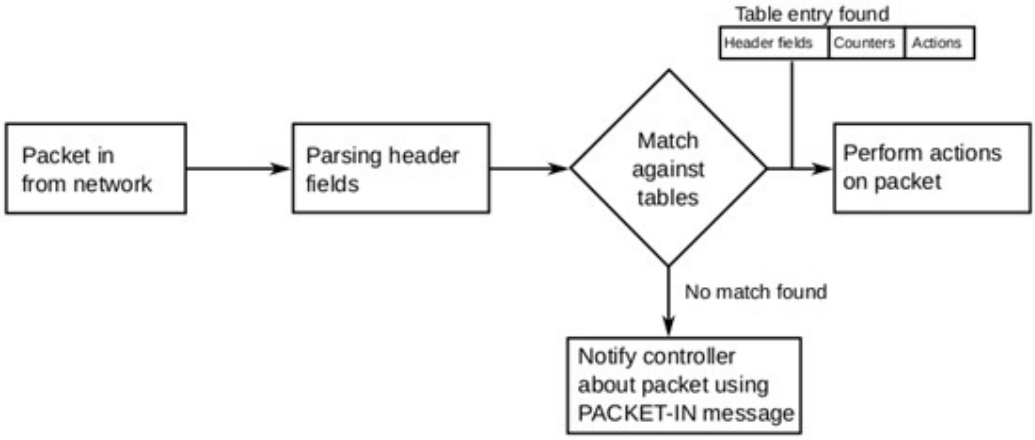


Figure 5: Προώθηση πακέτων από ένα switch με χρήση του Open Flow (Reference: [47])

Ο βασικός μηχανισμός προώθησης των πακέτων με το OpenFlow απεικονίζεται στο Σχήμα 5. Όταν ένα switch λαμβάνει ένα πακέτο, αναλύει την κεφαλίδα πακέτου. Αν βρεθεί ένας κανόνας του οποίου τα πεδία ταιριάζουν με τα πεδία του πακέτου, εφαρμόζεται αυτός ο κανόνας ενώ αν βρεθούν περισσότεροι απο ένας, εφαρμόζεται ο κανόνας με την μεγαλύτερη προτεραιότητα. Στη συνέχεια, το switch ενημερώνει τους αντίστοιχους μετρητές. Αν δεν υπάρχει καταχώρηση η οποία να ταιριάζει στο πακέτο τότε το switch στέλνει ένα packet-in μήνυμα στον ελεγκτή με ενσωματωμένο ολόκληρο το πακέτο, είτε με τα πρώτα bytes του αν υπάρχει δυνατότητα αποθηκευσης του πακέτου στον buffer του switch, όπως αναφέραμε παραπάνω. Ο ελεγκτής λαμβάνοντας το packet-in μήνυμα, είτε εγκαθιστά νέους κανόνες στο switch αποστέλλοντας τον κωδικό θέσης όπου είναι αποθηκευμένο το πακέτο μαζί με τις ενέργειες που πρέπει να εφαρμοστούν, είτε στέλνει μήνυμα Packet-Out.
[47]

Open Flow 1.1

Το Open Flow 1.1 [49] κυκλοφόρησε τον Φεβρουάριο του 2011. Περιέχει σημαντικές αλλαγές σε σχέση με το OpenFlow 1.0 καθώς η επεξεργασία των πακέτων λειτουργεί διαφορετικά. Οι δύο σημαντικές αλλαγές που εισάγονται είναι η σωλήνωση (pipeline) πολλαπλών πινάκων ροής και ο πίνακας ομάδας (group table).
Η αντιστοίχιση ενός πακέτου αρχίζει με τον πρώτο πίνακα ροής και συνεχίζεται στους επόμενους πίνακες ροής. Αν βρεθεί αντιστοίχιση σε κάποια εγγραφή του πίνακα ροής τότε θα εφαρμόστούν οι ενέργειές του. Αν δεν βρεθεί αντιστοίχιση τότε το πακέτο μπορεί (α) είτε να προωθηθεί στον ελεγκτή, (β) είτε να απορριφθεί ή (γ) να γίνει αντιστοίχιση με κάποια εγγραφή του επόμενου πίνακα ροής με την εντολή Goto. Οι πίνακες ροής του switch είναι αριθμημένοι σειριακά ξεκινώντας απο το 0 και ένας πίνακας ροής μπορεί να στείλει ένα πακέτο σε πίνακα με νούμερο

μεγαλύτερο απο το δικό του, εκτός από τον τελευταίο πίνακα ο οποίος δεν περιέχει την εντολή Goto. Η σωλήνωση (pipeline) χρησιμοποιείται ώστε τα πακέτα να περάσουν μεταξύ διαφορετικών πινάκων προκειμένου να αντιστοιχιθούν. Επίσης, παρέχει τη δυνατότητα να μεταφέρεται πληροφορία μεταξύ των διαφορετικών πινάκων με τη μορφή metadata.

Οι καταχωρίσεις του πίνακα ροής περιέχουν οδηγίες αντί για ενέργειες, όπως φαίνεται στο Σχήμα 6. Ο κατάλογος των πιθανών οδηγιών για το Open Flow 1.1 παρουσιάζεται στον Πίνακα 1. Η οδηγία «Apply-Actions» εφαρμόζει άμεσα ενέργειες στο πακέτο. Η εντολή «Write-Actions» προσθέτει τις συγκεκριμένες ενέργειες στο σύνολο των ενεργειών και επιτρέπει τη σταδιακή κατασκευή του συνόλου ενεργειών κατά την εκτέλεση του pipeline. Η εντολή «Clear-Actions» καθαρίζει το σύνολο των ενεργειών.

Η εντολή «Write-Metadata» ενημερώνει το πεδίο μεταδεδομένων. Τέλος, η εντολή «Goto» αναφέρεται στον πίνακα ροής όπου θα συνεχιστεί η διαδικασία αντιστοίχισης. Ο επόμενος πίνακας θα πρέπει να έχει υψηλότερο αναγνωριστικό από τον τρέχοντα πίνακα ώστε να αποφευχθούν οι βρόχοι. Σε περίπτωση που δεν έχει οριστεί η εντολή «Goto», τότε η επεξεργασία του pipeline σταματάει και το σύνολο ενεργειών εκτελείται στο πακέτο. [47]

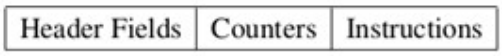


Figure 6: Εγγραφή του πίνακα ροής για το πρωτόκολλο 1.1 (Reference: [49]).

Instruction	Argument	Semantic
Apply-Actions	Action(s)	Applies actions immediately without adding them to the action set
Write-Actions	Action(s)	Merge the specified action(s) into the action set
Clear-Actions	-	Clear the action set
Write-Metadata	Metadata mask	Updates the metadata field
Goto-Table	Table ID	Perform matching on the next table

Table 1: Λίστα οδηγιών του πρωτοκόλλου Open Flow 1.1 (Reference: [47]).

Το Σχήμα 7 απεικονίζει τη διαδικασία επεξεργασίας των πακέτων του pipeline. Πριν ξεκινήσει το pipeline, το πεδίο μεταδεδομένων και η ενέργεια που έχει οριστεί για ένα πακέτο είναι κενά. Η διαδικασία αντιστοίχισης αρχίζει από τον πρώτο πίνακα ροής. Το πακέτο ταιριάζει με τους διαδοχικούς πίνακες ροής και από το καθένα επιλέγεται η καταχώριση με την υψηλότερη προτεραιότητα. Το pipeline τελειώνει όταν δεν υπάρχει καταχώρηση στον πίνακα ροής ή δεν έχει οριστεί εντολή «Goto» στην καταχώρηση του πίνακα ροής. Επιπλέον, το pipeline υποστηρίζει τον ορισμό σύνθετων μηχανισμών προώθησης και έτσι παρέχει μεγαλύτερη ευελιξία σε σχέση με τον μηχανισμό επεξερασίας της πρώτης έκδοσης του πρωτοκόλλου (OpenFlow 1.0).

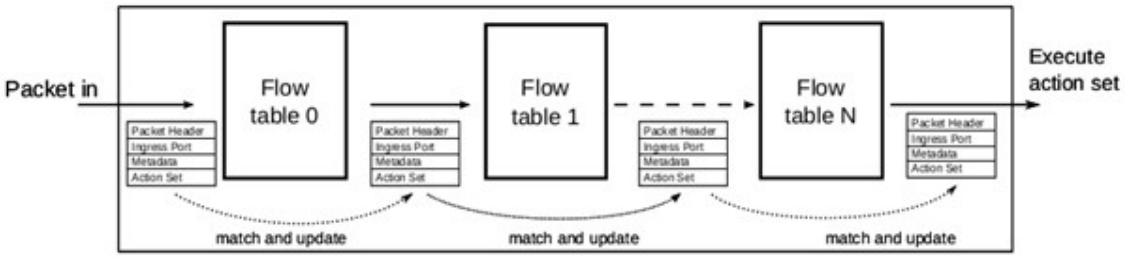


Figure 7: Open Flow pipeline (Reference: [47]).

Η δεύτερη αλλαγή αφορά την προσθήκη του πίνακα ομάδας (group table). Ο συγκεκριμένος πίνακας υποστηρίζει πιο σύνθετες συμπεριφορές προώθησης, οι οποίες εφαρμόζονται σε ένα σύνολο ροών.

Όπως παρατηρούμε και στο Σχήμα 8, ο group table περιλαμβάνει τα παρακάτω πεδία:

- Group Identifier: Ένας μη προσημασμένος ακέραιος αριθμός μήκους 32 bits, ο οποίος προσδιορίζει το κάθε group με μοναδικό τρόπο.
- Group Type: Προσδιορίζει τον τύπο του group.
- Counters: Είναι μετρητές οι οποίοι ανανεώνονται κάθε φορά που ένα πακέτο επεξεργάζεται από το αντίστοιχο group.
- Action Buckets: Είναι μια διατεταγμένη λίστα, από buckets με ενέργειες (action buckets). Κάθε bucket περιέχει ένα σύνολο ενεργειών που εκτελούνται κατά την επεξεργασία, καθώς και τις σχετικές με τις ενέργειες αυτές παραμέτρους.

Group Identifier	Group Type	Counters	Action Buckets

Figure 8: Εγγραφές του group table για το πρωτόκολλο 1.1 (Reference: [48]).

Στη συνέχεια θα αναφέρουμε τους διαφορετικούς τύπους Group Type:

- ALL: Εκτέλεση όλων των Action Buckets που ορίζονται στο αντίστοιχο πεδίο. Αυτό το group χρησιμοποιείται για προώθηση πακέτων, multicast ή broadcast. Για να το επιτύχουμε αυτό, κάθε πακέτο αντιγράφεται για κάθε bucket και τα αντίγραφα επεξεργάζονται από το αντίστοιχο bucket.
- SELECT: Τα πακέτα κατευθύνονται σε ένα μοναδικό bucket του Group, βάσει ενός αλγορίθμου επιλογής που υπολογίζεται από το ίδιο το switch.

- INDIRECT: Εκτέλεση του μοναδικού Action Bucket που είναι ορισμένο σε αυτό το Group. Αυτός ο τύπος είναι στην πράξη πανομοιότυπος με έναν all, στον οποίο είναι ορισμένο μόνο ένα Action bucket.
- FAST FAILOVER: Εκτέλεση του πρώτου ενεργού Action Bucket. Κάθε Action Bucket είναι συνδεδεμένο με ένα συγκεκριμένο port ή και group που ελέγχει την κατάσταση του bucket. Αυτός ο τύπος επιτρέπει στο switch να αλλάζει τον τρόπο προώθησης χωρίς να απαιτείται κάποια ενέργεια από τον controller.

[47]

Open Flow 1.2

Το Open Flow 1.2 [50] κυκλοφόρησε τον Δεκέμβριο του 2011 και υποστηρίζει το πρωτόκολλο IP version 6. Το Open Flow 1.2 μπορεί να ταιριάξει διευθύνσεις προέλευσης και προορισμού IPv6, την ετικέτα ροής, το traffic class και διάφορα πεδία του ICMPv6. Οι κατασκευαστές δικτυακών συσκευών έχουν νέες δυνατότητες ώστε να επεκτείνουν το OpenFlow μόνοι τους και να υποστηρίξουν πρόσθετες δυνατότητες αντιστοίχισης. Μια τιμή μήκους (type-length-value, TLV), η οποία ονομάζεται Open Flow Extensible Match (OXM), επιτρέπει σε κάποιον να ορίζει νέες καταχωρίσεις αντιστοίχισης με τρόπο επεκτάσιμο. Με τη συγκεκριμένη έκδοση, OpenFlow 1.2, ένα switch μπορεί ταυτόχρονα να συνδεθεί σε περισσότερους από έναν ελεγκτές, δηλαδή μπορεί να ρυθμιστεί ώστε να διαχειρίζεται από ένα σύνολο ελεγκτών. Το switch εκκινεί μια σύνδεση και οι ελεγκτές δέχονται τις προσπάθειες σύνδεσης. Ένας ελεγκτής ορίζεται ως master και προγραμματίζει το switch ενώ οι άλλοι ελεγκτές έχουν ρόλο slave. Ένας slave ελεγκτής μπορεί να προωθηθεί σε master, ενώ ο master να υποβιβαστεί στον ρόλο slave. Η συγκεκριμένη δυνατότητα μας επιτρέπει την εφαρμογή failover στους ελεγκτές. [47]

OpenFlow 1.3

Το Open Flow 1.3 [51] εισάγει νέα χαρακτηριστικά για τη λειτουργία (operations), την παρακολούθηση (monitoring) και τη διαχείριση (management) (OAM). Για το σκοπό αυτό, ένας πίνακας μετρητών ροής (meter table) προστίθεται στην αρχιτεκτονική του switch. Το Σχήμα 9 απεικονίζει τη δομή των καταχωρήσεων του πίνακα μετρητών. Ένα τέτοιος πίνακας, αποτελείται από αντίστοιχες εγγραφές (meter entries) οι οποίες ορίζουν μετρητές για κάθε ροή, δίνοντας την δυνατότητα υλοποίησης απλών ή συνθετότερων ενεργειών διασφάλισης ποιότητας (Quality of Service), όπως για παράδειγμα είναι ο περιορισμός της ροής της κίνησης (rate limiting). Κάθε τέτοιος δείκτης μετρά το ρυθμό μετάδοσης πακέτων που του έχει ανατεθεί, επιτρέποντας τον έλεγχο της ταχύτητας. Επισημαίνεται πως οι μετρητές ροής, είναι συνδεδεμένοι απευθείας με τις

εγγραφές, σε αντίθεση με τις ουρές (Queues) που είναι συνδεδεμένες με τις θύρες. Επίσης, ένας μετρητής ροής ελέγχει την συνολική ροή, από όλες τις εγγραφές που είναι συνδεδεμένες με αυτόν. Όσον αφορά την χρήση πολλαπλών μετρητών στον ίδιο πίνακα ροής, κάτι τέτοιο είναι εφικτό αλλά όχι για διαφορετικούς μετρητές στις ίδιες εγγραφές. Ωστόσο μπορούν να εφαρμοσθούν πολλαπλοί μετρητές στις ίδιες ροές πακέτων σε διαφορετικούς διαδοχικούς πίνακες ροής.
[47]

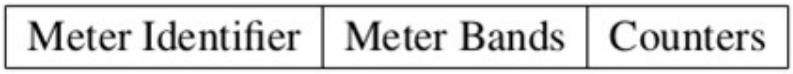


Figure 9: Εγγραφή του πίνακα meter (Reference: [48]).

Κάθε εγγραφή του πίνακα μετρητών αποτελείται από τα παρακάτω πεδία:

- Meter identifier: Προσδιορίζει μοναδικά την κάθε εγγραφή. Είναι ένας μη προσημασμένος ακέραιος αριθμός μήκους 32 bits.
- Meter bands: Πρόκειται για μη διατεταγμένη λίστα από meter bands, όπου η καθεμία ορίζει τον τρόπο επεξεργασίας των πακέτων που εμπίπτουν σε αυτές.
- Counters: Πρόκειται για μετρητές που ανανεώνονται όταν επεξεργάζονται πακέτα από τον αντίστοιχο μετρητή (meter).

Η υποστήριξη για πολλαπλούς ελεγκτές επεκτείνεται καθώς με το OpenFlow 1.2, μόνο η διαχείριση σφαλμάτων αντιμετωπίζεται από ένα σύστημα master / slave. Με το OpenFlow 1.3, μπορούν να χρησιμοποιηθούν βοηθητικές συνδέσεις για τη συμπλήρωση της σύνδεσης με τον κύριο ελεγκτή και το switch. Έτσι, μπορεί να επιτευχθεί καλύτερη εξισορρόπηση φορτίου στο επίπεδο ελέγχου. Επιπλέον, εισάγεται φιλτράρισμα συμβάντων ανά σύνδεση το οποίο επιτρέπει στους ελεγκτές να εγγράφουν μόνο σε τύπους μηνυμάτων που τους ενδιαφέρουν. Για παράδειγμα, ένας ελεγκτής που είναι υπεύθυνος για τη συλλογή στατιστικών στοιχείων μπορεί να συνδεθεί ως βοηθητικός ελεγκτής σε ένα switch και να καταγράφει μόνο στα στατιστικά στοιχεία που παράγει. Το OpenFlow 1.3 υποστηρίζει τις κεφαλίδες επεκτάσεων IPv6. Αυτό περιλαμβάνει, π.χ., αντιστοίχιση στην κεφαλίδα IPv6 κρυπτογραφημένου ωφέλιμου φορτίου ασφαλείας (ESP), κεφαλίδα ελέγχου ταυτότητας IPv6 ή επικεφαλίδα IPv6 hop-by-hop. Επιπλέον, προστίθεται υποστήριξη για Provider Backbone Bridge (PBB), καθώς και άλλες βελτιώσεις δευτερευόντων πρωτοκόλλων. [47]

OpenFlow 1.4

Το OpenFlow 1.4 [52] κυκλοφόρησε τον Οκτώβριο του 2013. Το ONF βελτίωσε την υποστήριξη για το OpenFlow Extensible Match (OXM). Οι δομές TLV για θύρες, πίνακες και ουρές προστίθενται στο πρωτόκολλο και τα hard-coded μέρη από προηγούμενες εκδόσεις αντικαθίστανται από τις νέες δομές TLV και η διαμόρφωση των εικονικών θυρών είναι τώρα εφικτή. Επιπλέον, οι ελεγκτές μπορούν να στέλνουν μηνύματα ελέγχου με μία δέσμη μηνυμάτων στα switches. Επίσης, περιλαμβάνονται μικρές βελτιώσεις των group tables και στα χαρακτηριστικά παρακολούθησης. [47]

OpenFlow 1.5

Στις προηγούμενες εκδόσεις του πρωτοκόλλου, η επεξεργασία των πακέτων γινόταν στην εισερχόμενη θύρα. Στην έκδοση 1.5 του πρωτοκόλλου OpenFlow χρησιμοποιούνται οι πίνακες egress σύμφωνα με τους οποίους υπάρχει η δυνατότητα η επεξεργασία να γίνει και στην εξερχόμενη θύρα. Η διαδικασία του pipeline γίνεται σε δύο στάδια, την επεξεργασία εισόδου (ingress processing) και την επεξεργασία εξόδου (egress processing). Ο διαχωρισμός των δύο σταδίων υποδεικνύεται από τον πρώτο πίνακα egress. Όλοι οι πίνακες με αριθμό μικρότερο από τον πρώτο πίνακα egress χρησιμοποιούνται ως πίνακες ingress και κανένας πίνακας με αριθμό μεγαλύτερο ή ίσο από τον πρώτο πίνακα egress δεν μπορεί να χρησιμοποιηθεί ως ingress table.
Η διαδικασία του pipeline ξεκινά με την επεξεργασία εισόδου στον πρώτο πίνακα ροής. Το πακέτο θα πρέπει να αντιστοιχηθεί με κάποια από τις εγγραφές του πρώτου πίνακα. Οι υπόλοιποι πίνακες ingress μπορεί να χρησιμοποιηθούν ανάλογα με το αποτέλεσμα της αντιστοίχισης του πρώτου πίνακα. Εάν το αποτέλεσμα είναι να προωθηθεί το πακέτα σε κάποια από τις θύρες εξόδου, το OpenFlow switch μπορεί να πραγματοποιήσει επεξεργασία εξόδου στα πλαίσια της θύρας εξόδου. Η επεξεργασία εξόδου είναι προαιρετική, αφού ένα switch μπορεί να μην υποστηρίζει καθόλου egress tables. Εάν δεν υπάρχει egress table τότε το πακέτο προωθείται από το switch. Εάν υπάρχει egress table, το πακέτο μπορεί να αντιστοιχηθεί με τις εγγραφές αυτού του πίνακα ή να χρειαστούν και άλλα egress tables, ανάλογα με το αποτέλεσμα της αντιστοίχισης.

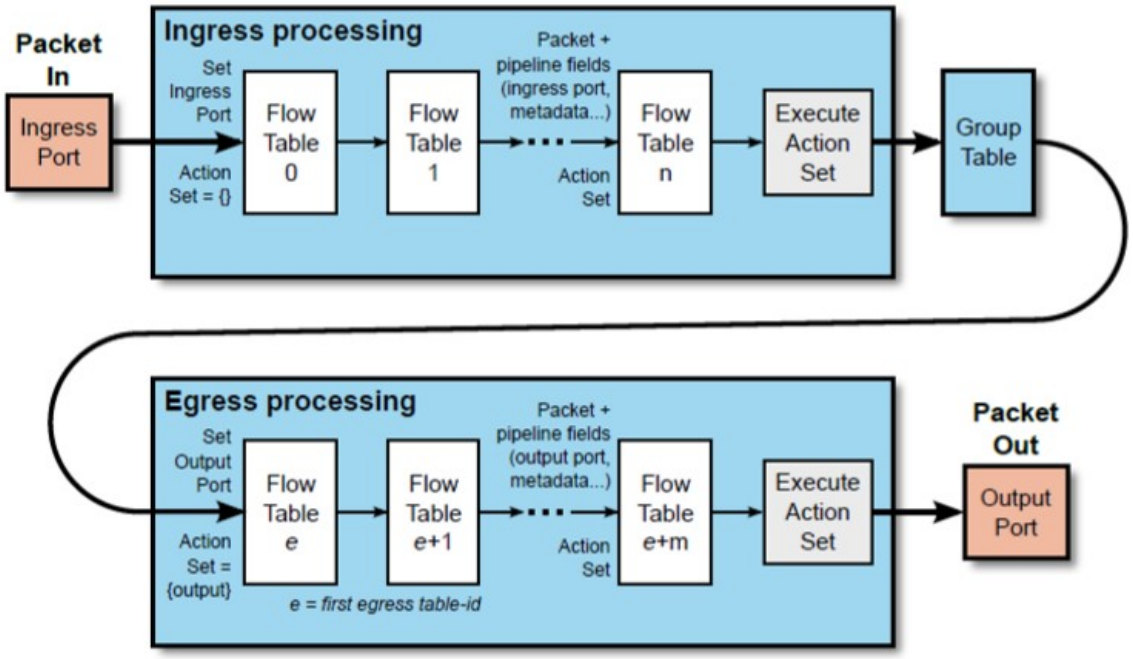


Figure 10: Επεξεργασία πακέτων στο πρωτόκολλο Open Flow 1.5 (Reference: [53]).

Σε όλες τις προηγούμενες εκδόσεις, όλα τα πακέτα έπρεπε να είναι Ethernet. Η έκδοση 1.5 του OpenFlow υποστηρίζει και άλλα ήδη πακέτων, όπως τα PPP πακέτα. Ένα νέο πεδίο OXM pipeline χαρακτηρίζει το είδος κάθε πακέτου.

Επίσης, εισάγονται τα Openflow eXtensible Statistics (OXS) για να κωδικοποιήσουν τα στατιστικά των εγγραφών. Τα πεδία που περιλαμβάνουν είναι η διάρκεια εγγραφής, το πλήθος εγγραφών, το πλήθος πακέτων, και σύνολο απο bytes.

Τα στατιστικά αποστέλνονται στον controller όταν φτάσουν κάποια όρια αριθμού στατιστικών. Επίσης έχουμε μια νέα εντολή, την OFPIT_STAT_TRIGGER η οποία καθορίζει ένα σύνολο στατιστικών ορίων χρησιμοποιώντας τα OXS.

Επίσης, υπάρχει η δυνατότητα να έχουμε αντιστοίχιση στα TCP flags SYN, ACK και FIN ώστε να εντοπιστεί η αρχή και το τέλος μιας TCP σύνδεσης. [53]

Open Flow Controllers

Στην τεχνολογία SDN, ο ελεγκτής είναι το βασικό στοιχείο που επιτρέπει τη διαχείριση του δικτύου μέσω των υποδομών δικτύωσης. Παρέχει abstractions για τη σύνδεση και επικοινωνία με τις συσκευές προώθησης, την πρόσβαση σε πόρους, τη δημιουργία και τη διατήρηση των ρυθμίσεων των συσκευών και την προώθηση πολιτικών.

Από την οπτική γωνία της αρχιτεκτονικής του συστήματος, οι ελεγκτές SDN μπορούν να χωριστούν σε δύο κύριες ομάδες: τους κεντρικούς ελεγκτές και τους κατανεμημένους ελεγκτές.

Όπως φαίνεται στο σχήμα 11 (a), ένας κεντρικός ελεγκτής είναι μια ενιαία οντότητα που διαχειρίζεται όλες τις συσκευές προώθησης του δικτύου.

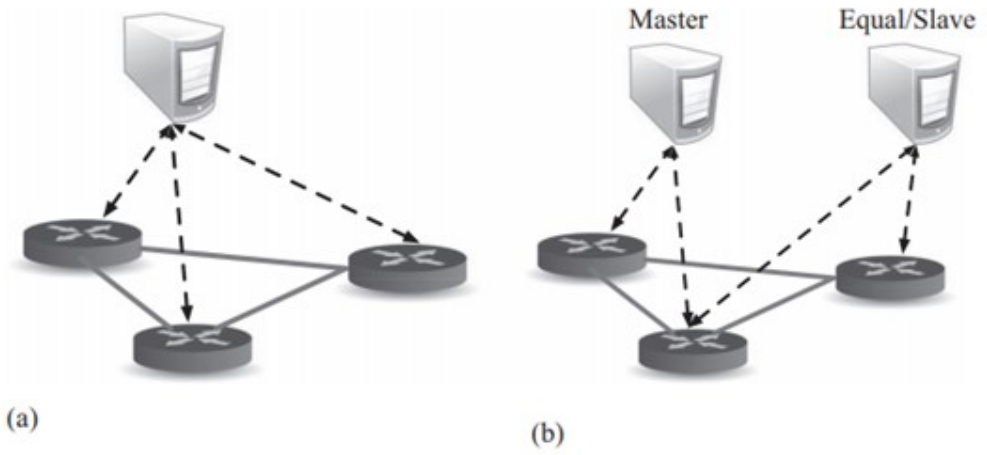


Figure 11: Αρχιτεκτονικές ελεγκτών SDN. (a) κεντρικός ελεγκτής SDN (b) κατανεμηνένοι ελεγκτές SDN (Reference: [12]).

Το NOX [2] είναι ο πρώτος προτεινόμενος ελεγκτής SDN που υποστηρίζει το πρωτόκολλο Open Flow. Η έκδοσή του που είναι γραμμένη σε Python (POX), διαδραματίζει σημαντικό ρόλο για τις εφαρμογές SDN.

Για την ικανοποίηση των συνεχώς αυξανόμενων απαιτήσεων απόδοσης, ειδικά για τα enterprise δίκτυα και τα data centers προτείνεται η χρήση περισσότερων κεντρικών ελεγκτών για την ενίσχυση της απόδοσης.

Ένα παράδειγμα είναι το Beacon το οποίο έχει υιοθετηθεί ευρέως τόσο σε ερευνητικά πειράματα όσο και στη βιομηχανία, όπως η Amazon, για την υψηλή απόδοση, την επεκτασιμότητα και τη σταθερότητά του. Η επιτυχία του ανήκει στην αρθρωτή και cross-platform αρχιτεκτονική του, καθώς και στο εύκολο στη χρήση μοντέλο προγραμματισμού και στις σταθερές διεπαφές χρήστη.

Οι κεντρικοί ελεγκτές συνέβαλαν στην ανάπτυξης, εξέλιξη και εφαρμογή της τεχνολογίας SDN στα πρώιμα στάδιά της. Ωστόσο, μπορεί να έχουν περιορισμούς κλιμάκωσης (scaling), γεγονός που εμποδίζει την υιοθέτησή τους για τη διαχείριση μεγάλου αριθμού στοιχείων επιπέδου δεδομένων.

Πρώτον, οι πόροι σε μια ενιαία οντότητα είναι περιορισμένοι. Δεύτερον, σε ένα δίκτυο μεγάλης κλίμακας, ανεξάρτητα από το σημείο όπου θα εγκατασταθεί ο ελεγκτής, υπάρχουν ορισμένες συσκευές προώθησης που παρουσιάζουν μεγάλη καθυστέριση, ώστε να διαμορφωθούν και διαχειριστούν σε πραγματικό χρόνο. Τελευταίο αλλά όχι λιγότερο σημαντικό, είναι ότι ο κεντρικός ελεγκτής είναι single point of failure και στόχος για την ασφάλεια του δικτύου.

Αντίθετα, οι κατανεμημένοι ελεγκτές θα μπορούσαν να είναι πιο κλιμακωτοί (scalable) για να ικανοποιήσουν τις πιθανές απαιτήσεις τόσο των μικρών όσο και των μεγάλων δικτύων. Όπως φαίνεται στο Σχήμα 11, ένας κατανεμημένος ελεγκτής αποτελείται από ένα σύνολο φυσικά κατανεμημένων στοιχείων, τα οποία θα μπορούσαν να είναι πιο ανθεκτικά σε διαφορετικά είδη λογικών και φυσικών βλαβών. Ωστόσο, δεδομένου ότι οποιοσδήποτε κόμβος - ελεγκτής εντός ενός κατανεμημένου ελεγκτή πρέπει να διατηρεί τουλάχιστον μία σύνδεση με μια συσκευή προώθησης, είναι σημαντικό να εξισορροπηθεί το φορτίο μεταξύ όλων των κόμβων ελέγχου. Για παράδειγμα, η ElastiCon προτείνει μια σειρά καινοτόμων μηχανισμών για την παρακολούθηση του φορτίου σε κάθε κόμβο - ελεγκτή, τη βελτιστοποίηση της κατανομής φορτίου σύμφωνα με την ανάλυση της κατάστασης και τη μετεγκατάσταση των συσκευών προώθησης από τους πολύ φορτωμένους κόμβους ελεγκτών σε άλλους ελαφρώς φορτωμένους.

Ωστόσο, οι αποφάσεις διανομής του φόρτου γίνονται πάντοτε βάσει ενός προκαθορισμένου ορίου, το οποίο δεν μπορεί να διασφαλιστεί ως το βέλτιστο καθώς το δίκτυο αναπτύσσεται.

Ένα άλλο ζήτημα των κατανεμημένων ελεγκτών είναι η συνέπεια (consistency). Οι περισσότεροι υπάρχοντες ελεγκτές, όπως ο DIStributed SDN COntroller (DISCO), έχουν χαμηλή συνέπεια. Συγκεκριμένα, σε αυτούς τους ελεγκτές, διαφορετικοί κόμβοι μπορούν να μάθουν διαφορετικές τιμές της ίδιας ιδιότητας κάποια στιγμή, επειδή οι ενημερώσεις δεδομένων δεν μπορούν να εξαπλωθούν σε όλους τους κόμβους αμέσως. Επί του παρόντος, μόνο μερικές προτάσεις όπως το Onix και το SMaRtLight παρέχουν σχετικά ισχυρή συνέπεια, η οποία εξασφαλίζει τουλάχιστον ότι όλοι οι κόμβοι διαβάσουν την τελευταία τιμή της ίδιας ιδιότητας μετά από μια εγγραφή. Όμως το κόστος για να το επιτύχουμε είναι η απόδοση. [12]

Στον παρακάτω πίνακα παρουσιάζονται οι διαθέσιμοι open source κεντρικοί ελεγκτές.

Ελεγκτής	Γλώσσα Προγραμματισμού
NOX	C++
POX	Python
Beacon	Java
Floodlight	Java
Maestro	Java
NodeFlow	Javascript
Thema	C και Ruby
OpenDaylight	Java

Table 2: Διαθέσιμοι OpenFlow ελεγκτές ανοικτού κώδικα (Reference: [47]).

Εξομοιωτές SDN

Το Mininet είναι ένας εξομοιωτής δικτύου που δημιουργεί ένα δίκτυο εικονικών υπολογιστών (hosts), switches, ελεγκτές και συνδέσμους (links). Οι υπολογιστές του Mininet χρησιμοποιούν τυπικό λογισμικό δικτύου Linux και τα switches υποστηρίζουν το OpenFlow.

Το Mininet χρησιμοποιεί process-based virtualization για να τρέξει πολλούς (έχουν ξεκινήσει με επιτυχία έως και 4096) hosts και switches πάνω από ένα πυρήνα λειτουργικού συστήματος. Από την έκδοση 2.2.26, το Linux υποστηρίξει τα network namespaces, ένα ελαφρύ χαρακτηριστικό γνώρισμα virtualization το οποίο παρέχει μεμονωμένες διεργασίες με ξεχωριστές διεπαφές δικτύου, πίνακες δρομολόγησης και πίνακες ARP ώστε να διαχωρίζει τις διασυνδέσεις δικτύου, τους πίνακες δρομολόγησης και τους πίνακες ARP των διαφορετικών virtual hosts. Τα virtual switch του Mininet είναι συμβατά με το πρωτόκολλο OpenFlow και ονομάζονται Open vSwitch. Ο πυρήνας του Linux παρέχει εικονικά ζεύγη Ethernet (veth) για να επιτευχθούν οι διασυνδέσεις μεταξύ των virtual hosts και των virtual switches. Σε ένα δίκτυο που προσομοιώνεται στο Mininet, τα virtual switches πρέπει να συνδεθούν με έναν OpenFlow ελεγκτή ο οποίος μπορεί να εκτελείται σε έναν virtual host ή σε κάποιο εξωτερικό μηχάνημα. Το Mininet διαθέτει εγκατεστημένο τον ελεγκτή POX.

Το Mininet:

- Προσφέρει ένα απλό και φθηνό περιβάλλον δικτύου για την ανάπτυξη εφαρμογών OpenFlow,
- Προσφέρει τη δυνατότητα σε πολλούς προγραμματιστές ταυτόχρονα να εργάζονται ανεξάρτητα στην ίδια τοπολογία,
- Επιτρέπει δοκιμές με πολύπλοκες τοπολογίες, χωρίς να απαιτείται η σύνδεση φυσικού δικτύου,

- Παρέχει CLI - Command LIne που είναι πλήρως ενημερωμένο για την τοπολογία και το OpenFlow, για τον εντοπισμό σφαλμάτων ή τη διεξαγωγή δοκιμών σε όλο το δίκτυο,
- Επιτρέπει Custom τοπολογίες και περιλαμβάνει ένα βασικό σύνολο παραμετροποιημένων τοπολογιών,
- Μπορεί να χρησιμοποιηθεί χωρίς να απαιτείται προγραμματισμός, αλλά
- Παρέχει ένα απλό και επεκτάσιμο Python API για δημιουργία δικτύων και πειραματισμό.

Στους περιορισμούς, τα δίκτυα που βασίζονται στο Mininet δεν μπορούν (αυτή τη στιγμή) να υπερβούν τη διαθέσιμη CPU ή εύρος ζώνης σε ένα server. Επίσης, το Mininet δεν μπορεί (αυτή τη στιγμή) να εκτελέσει switch ή εφαρμογές OpenFlow που δεν είναι συμβατα με Linux, κάτι που στην πράξη δεν είναι σημαντικό ζήτημα. [13]

Τεχνολογία Παραπλάνησης (Deception Technology)

Ο σκοπός της τεχνολογίας παραπλάνησης είναι να αποτρέψουμε έναν επιτιθέμενο που έχει καταφέρει να διεισδύσει σε ένα δίκτυο να προκαλέσει σημαντικές ζημιές. Αυτή η τεχνολογία λειτουργεί με τη δημιουργία παγίδων ή παραπλανητικών δολωμάτων που μιμούνται τους κανονικούς υπολογιστές μιας υποδομής. Τα δολώματα μπορούν να τρέξουν σε ένα εικονικό ή πραγματικό περιβάλλον και έχουν σχεδιαστεί για να εξαπατήσουν τους επιτιθέμενους κάνοντας τους πιστέψουν ότι έχουν ανακαλύψει έναν τρόπο να αυξήσουν τα προνόμια τους (privilege escalation) και να κλέψουν διάφορα διαπιστευτήρια (credentials). Μόλις ενεργοποιηθεί μια παγίδα, μεταδίδονται ειδοποιήσεις σε ένα κεντρικό deception server όπου καταγράφεται το δόλωμα και ο τρόπος επίθεσης που χρησιμοποίησε ο επιτιθέμενος.

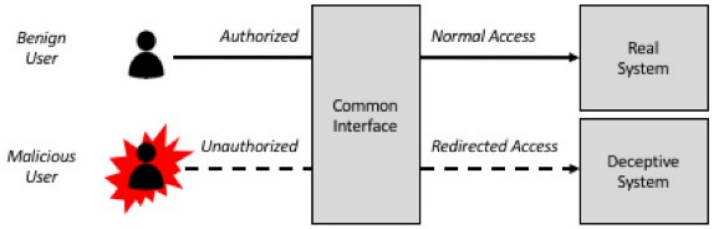


Figure 12: Deception Technology (Reference: [14]).

Στη συνέχεια θα δούμε κάποιους λόγους για να χρησιμοποιήσουμε Deception Technology.

Ανίχνευση μετά από παραβίαση

Κανένας μηχανισμός ασφάλειας δεν μπορεί να σταματήσει όλες τις επιθέσεις σε ένα δίκτυο, αλλά η τεχνολογία παραπλάνησης προσφέρει στους επιτιθέμενους μια ψεύτικη αίσθηση ασφάλειας, κάνοντάς τους να πιστεύουν ότι έχουν κερδίσει ένα πάτημα στο δίκτυό. Από εκεί μπορούμε να παρακολουθούμε και να καταγράφουμε τη συμπεριφορά τους με ασφάλεια, γνωρίζοντας ότι δεν μπορούν να προκαλέσουν ζημιά στα συστήματά μας. Οι πληροφορίες που θα καταγραφούν σχετικά με τη συμπεριφορά των επιτιθεμένων και τις τεχνικές τους μπορούν να χρησιμοποιηθούν για την ασφαλέστερη προστασία του δικτύου μας από την επίθεση.

Μειωμένα False Positives και μειωμένος κίνδυνος

Τα False Positives μπορούν να παρεμποδίσουν τις προσπάθειες για ασφαλείας και να εξαντλήσουν πόρους, χωρίς καν να αναλυθούν. Ο υπερβολικός θόρυβος από False Positives θα μπορούσε να μας οδηγήσει να αγνοήσουμε μια κανονική απειλή. Η τεχνολογία παραπλάνησης μειώνει το θόρυβο με λιγότερα false positives με alerts γεμάτα με χρήσιμα δεδομένα.

Η συγκεκριμένη τεχνολογία είναι επίσης χαμηλού κινδύνου, καθώς δεν υπάρχει κίνδυνος για τα δεδομένα ή επιπτώσεις στους πόρους της επιχειρήσεις. Όταν ένας επιτιθέμενος αποκτά πρόσβαση ή επιχειρεί να χρησιμοποιήσει μέρος ενός μηχανισμού deception, δημιουργείται μια ειδοποίηση προς τους διαχειριστές.

Κλιμάκωση και αυτοματοποίηση

Ενώ οι απειλές για τα εταιρικά δίκτυα και τα δεδομένα αποτελούν ένα καθημερινό πρόβλημα, οι ομάδες ασφαλείας σπάνια αυξάνουν τον προϋπολογισμό τους για να αντιμετωπίσουν τον κατακλυσμό νέων απειλών. Για το λόγο αυτό, η τεχνολογία παραπλάνησης μπορεί να είναι μια πολύ ευπρόσδεκτη λύση, καθώς οι αυτοματοποιημένες ειδοποιήσεις εξαλείφουν την ανάγκη για χειροκίνητη παρέμβαση, ενώ ο σχεδιασμός της τεχνολογίας επιτρέπει την εύκολη κλιμάκωση.

Μία από τις σημαντικότερες απαιτήσεις για την επιτυχή υλοποίηση της τεχνολογίας παραπλάνησης είναι ότι πρέπει να παραμείνει δυσδιάκριτη στον επιτιθέμενο. Εάν ο επιτιθέμενος υποπτευθεί ότι εξαπατάται, θα κάνει ότι μπορεί για να αποφύγει τις παγίδες και θα αυξήσει τις προσπάθειές του ώστε να φτάσει στα πραγματικά μηχανήματα. [14]

Τεχνικές Παραπλάνησης (Deception Techniques)

Τόσο ο ακαδημαϊκός όσο και ο βιομηχανικός τομέας προσπαθούν να αναπτύξουν αξιόπιστα δίκτυα υπολογιστών αξιοποιώντας διάφορες τεχνικές παραπλάνησης ώστε, εκ των προτέρων, να αναγνωρίζουν και να μετριάζουν πιθανές επιθέσεις στον κυβερνοχώρο. Όσο οι επιθέσεις στον κυβερνοχώρο γίνονται πιο εξελιγμένες, οι τεχνικές αμυντικής παραπλάνησης γίνονται όλο και πιο δημοφιλείς, δεδομένου ότι διευκολύνουν τη διαχείριση και την παρακολούθηση των δικτύων.

Οι **πέντε κύριες ιδιότητες** [15] που πρέπει να έχει ένας αμυντικός μηχανισμός παραπλάνησης είναι:

Ιδιότητα 1: Να αυξάνει τον φόρτο εργασίας του εισβολέα,

Ιδιότητα 2: Να επιτρέπει στους αμυνόμενους να παρακολουθούν τις επιθέσεις και να ανταποκριθούν πριν οι αντίπαλοι πετύχουν το σκοπό τους,

Ιδιότητα 3: Να εξαντλήσει τους πόρους του αντιπάλου,

Ιδιότητα 4: Να αυξάνει την πολυπλοκότητα που απαιτείται για την επίθεση,

Ιδιότητα 5: Να αυξάνει την αβεβαιότητα του εισβολέα.

Η ασφάλεια των πληροφοριών απαιτεί διάφορα τεχνικά και οργανωτικά μέτρα και παρόλο που είναι δύσκολο να εφαρμοστούν διαφορετικές τεχνικές αμυντικής παραπλάνησης για την πλήρη προστασία υποδομών ΤΠΕ, έχουν γίνει προσπάθειες ώστε να υιοθετηθούν τέτοιες τεχνολογίες στα σύγχρονα συστήματα.

Ο D.Fraunholz διεξήγαγε μια ολοκληρωμένη έρευνα σχετικά με τις τεχνικές παραπλάνησης [16]. Στην εργασία του ταξινόμησε διάφορους μηχανισμούς ασφαλείας σύμφωνα με τα αντικείμενα (assets) που προσπαθούν να προστατεύσουν.

Έχουν αναγνωριστεί τρία επίπεδα εφαρμογής των τεχνικών παραπλάνησης, (α) το Δίκτυο, (β) το Σύστημα και (γ) το επίπεδο δεδομένων. Για κάθε επίπεδο εντοπίστηκαν μηχανισμοί ασφάλειας που θα μπορούσαν να χρησιμοποιηθούν σε υποδομές πληροφορικής για προστασία των assets.

Επιπλέον, έχουν διαχωριστεί οι τεχνολογίες ασφάλειας που βασίζονται στην παραπλάνηση για κάθε επίπεδο. Ο X.Han στο [17] έχει εξετάσει τις τεχνικές παραπλάνησης στην Ασφάλεια Υπολογιστών και τις έχει κατηγοριοποιήσει σύμφωνα με τη μονάδα παραπλάνησης (unit of deception), το επίπεδο όπου εφαρμόζεται η παραπλάνηση, το στόχο της παραπλάνησης και τον τρόπο ανάπτυξής τους. Σε αυτή την ταξινόμηση υπάρχουν τέσσερα επίπεδα, (α) το Δίκτυο, (β) το Σύστημα, (γ) οι Εφαρμογές και (δ) το επίπεδο δεδομένων.

Αυτή η εργασία επικεντρώνεται στις τεχνολογίες παραπλάνησης με βάση το δίκτυο, προκειμένου να εξεταστεί η εφαρμογή τους στην τεχνολογία SDN. Σύμφωνα με το [17], οι τεχνολογίες παραπλάνησης με βάση το δίκτυο στοχεύουν στην άμβλυνση τριών κατηγοριών απειλών.

- network fingerprinting,

- υποκλοπές (eaves dropping) και
- διείσδυση (infiltration) και διάδοση της επίθεσης (attack propagation).

Στη βιβλιογραφία, έχουν εντοπιστεί οι παρακάτω δέκα τεχνικές:

- Network Tarpit: Αυτή η τεχνική εστιάζει στις sticky συνδέσεις που αποσκοπούν στην επιβράδυνση ή το σταμάτημα μιας αυτοματοποιημένης σάρωσης του δικτύου με σκοπό να μπερδέψει τους επιτιθέμενους.
- Traffic forging: Αυτή η τεχνική αυξάνει τη ροή της κίνησης στο δίκτυο προκειμένου να επιβραδύνει τις ενέργειες του αντιπάλου.
- Deceptive topology: Αυτή η τεχνική στοχεύει να παραμορφώσει την τοπολογία του δικτύου μέσω του traffic forging για να καθυστερήσει τον επιτιθέμενο.
- OS obfuscation: Μέσω αυτής της τεχνικής, δημιουργείται μια μίμηση της δικτυακής συμπεριφοράς πλαστών λειτουργικών συστημάτων για να εξαπατηθούν οι επιτιθέμενοι.
- Honeytokens: Τα honeytokens αποτελούνται από passwords, παραμέτρους URL, database honeytokens και honey permissions. Μέσω αυτών, θα μπορούσαν να χρησιμοποιηθούν πολλές μέθοδοι παραπλάνησης, όπως η ανταλλαγή εικονικών credentials μέσω του δικτύου, προκειμένου να τα εντοπίσουν και να τα χρησιμοποιήσουν οι εισβολείς και να εκθέσουν την κακόβουλη δραστηριότητά τους.
- Deceptive attack graphs: Αυτή η τεχνική χρησιμοποιεί attack graph representations, ώστε να αποσπάσει την προσοχή των αντιπάλων από του πραγματικούς στόχους.
- Deceptive simulation: Αυτή η τεχνική έχει χρησιμοποιηθεί και από βιομηχανικά συστήματα ελέγχου και η συγκεκριμένη προσομοίωση επιτρέπει την παρακολούθηση της τοπολογίας του δικτύου και τη δημιουργία ψευδών στόχων ώστε να εξαπατήσουν τους επιτιθέμενους.
- Decoy services: Ο αμυνόμενος μοιράζει ψεύτικα μηνύματα πρωτοκόλλου, καθυστερεί τις απαντήσεις και δημιουργεί μηνύματα σφάλματος ώστε να καθυστερήσει τον εισβολέα.
- Moving Target Defense: Υπάρχουν πολλές τεχνικές που θα μπορούσαν να περιγραφούν ως MTD, όπως η Address Space Layout Randomization- ASLR, η Instruction Set Randomization και η Code Sequence Randomization. Ο G.I.Cai στο [18] περιγράφει την MTD ως ασύμμετρη κατάσταση η οποία αλλάζει συνεχώς την επιφάνεια επίθεσης των προστατευμένων συστημάτων.
- Honeypots: Αυτή η τεχνική χρησιμοποιείται ευρέως ως αμυντική τεχνολογία παραπλάνησης προκειμένου να ενθαρρύνει και να εξαπατήσει τους αντιπάλους ώστε να εκμεταλλευτούν κάποια ευπάθεια. Για το σκοπό αυτό, σύμφωνα με το [19], τα honeypots είναι κατασκευασμένα να εκθέτουν τις ευπάθειές τους στους επιτιθέμενους εξομοιώνοντας ή προσομοιάζοντας συστήματα όπως για παράδειγμα Βάσεις Δεδομένων, Servers, File Systems και διάφορες υπηρεσίες, όπως είναι ο έλεγχος ταυτότητας (authentication).

Στον παρακάτω πίνακα παρουσιάζονται οι υπάρχουσες εργασίες για την αμυντική παραπλάνηση στα δίκτυα υπολογιστών σύμφωνα με την τεχνική που χρησιμοποιούν.

Reference	Technique
[20], [21], [22]	Network Tarpit
[23]	Traffic forging
[24]	Deceptive Technology
[25]	OS obfuscation
[26], [27]	Honeytokens
[28], [29]	Deceptive attack graph
[30]	Decoy services
[31]	Deceptive simulation
[32]	Moving Target Defense
[33]	Honeypots

Table 3: Κατηγορίες αμυντικής παραπλάνησης στα δίκτυα υπολογιστών.

Τεχνικές αμυντικής παραπλάνησης σε SDN (Defensive deception techniques on SDN)

Οι τεχνικές αμυντικής παραπλάνησης στα SDN δεν έχουν αναπτυχθεί πλήρως, δεδομένου ότι η τεχνολογία SDN είναι αρκετά πρόσφατη. Τα τελευταία χρόνια τόσο η βιομηχανία όσο και η ακαδημαϊκή κοινότητα προσπαθούν να εξελίξουν την τεχνολογία SDN σε διάφορους τομείς. Έχουν καταβληθεί προσπάθειες για τη σχεδίαση συστημάτων, την κλιμάκωση, την ενίσχυση της ανθεκτικότητας και την ταυτόχρονη βελτίωση της ασφάλειας και της αξιοπιστίας του SDN. Παρόλο που η τεχνολογία SDN είναι μια νέα τάση, οι αμυντικές τεχνικές παραπλάνησης για τέτοια δίκτυα δεν έχουν αναλυθεί λεπτομερώς. Όπως φαίνεται στον παρακάτω πίνακα, μόνο τέσσερις από τις δέκα τεχνικές παραπλάνησης έχουν μελετηθεί με την έννοια του SDN.

SDN	Defensive Deception Techniques
[34]	Deceptive Technologies
[35]	Decoy Services
[36][37][33]	Honeypots, Honeytokens
[38][39][40][41][42][43][44][45][46]	Moving Target Defense

Table 4: Τεχνικές παραπλάνησης στα SDN

Για να κατανοήσουμε καλύτερα την ενσωμάτωση των αμυντικών τεχνικών παραπλάνησης στο SDN, θα αναλύσουμε ξεχωριστά κάθε μια από τις συγκεκριμένες τεχνικές. Για το σκοπό αυτό, θα αναλυθούν οι στόχοι και οι λειτουργίες του SDN ως μέρος μιας αμυντικής τεχνικής παραπλάνησης.

Σχετικές Εργασίες

Δεν έχει γίνει καμία εργασία προκειμένου να ερευνηθούν οι υπάρχουσες τεχνικές αμυντικής παραπλάνησης με τη χρήση SDN. Ωστόσο, διαφορετικές εργασίες στη βιβλιογραφία έχουν εξετάσει την εφαρμογή τεχνικών αμυντικής παραπλάνησης σε παραδοσιακά δίκτυα ή συστήματα. Ο K.Hoffman στο [7] προτείνει ένα πλαίσιο για τον εντοπισμό πιθανών επιθέσεων και μηχανισμούς άμυνας στα συστήματα repudiation. Επιπλέον, ο S.Jajodia στο [8] έχει αναλύσει την εφαρμογή της τεχνικής Moving Target Defense (MTD) για την προστασία σύγχρονων δικτύων υπολογιστών. Παρ' όλα αυτά δεν έχουν εξετάσει την εφαρμογή της MTD στα SDN αξιοποιώντας τις δυνατότητές τους. Ο C.Lei στο [9] διεξήγαγε μια έρευνα σχετικά με διαφορετικές τεχνικές MTD. Μέσα από την εργασία τους, έχουν αναλύσει τις αρχές σχεδιασμού και την αρχιτεκτονική του συστήματος της τεχνικής MTD. Ακολουθώντας την ίδια κατεύθυνση, ο B. C. Ward στο [10] παρέχει μια επισκόπηση των διαφορετικών τεχνικών cyber-MTD, των μοντέλων απειλών τους (threat models)

και των τεχνικών τους λεπτομερειών. Ωστόσο, κανένα από τα προηγούμενα έργα δεν έχει επικεντρωθεί την εφαρμογή του MTD σε SDN.

Επιπλέον, διεξήχθησαν διαφορετικές έρευνες για να εξεταστούν οι εφαρμογές και οι δυνατότητες του SDN. Συγκεκριμένα, ο S.Rowshanrad στο [11] αναλύει τις διαφορετικές εφαρμογές SDN και τις διαφορετικές southbound διασυνδέσεις. Αναφέρουν την πιθανή εφαρμογή του SDN σε cloud, ασύρματα και κινητά δίκτυα και καθορίζουν τα οφέλη ασφάλειας που απορρέουν από τέτοιες εφαρμογές. Επιπλέον, διεξήχθη ολοκληρωμένη έρευνα για το SDN από τον D.Kreutz στο [3]. Συγκεκριμένα, οι συγγραφείς αναλύουν λεπτομερώς την αρχιτεκτονική SDN και παρέχουν τις απαραίτητες λειτουργίες και απαιτήσεις που καθιστούν αυτή την υποδομή σημαντική για τα μελλοντικά δίκτυα. Αν και υπάρχουν έρευνες για την τεχνολογία SDN, κανείς δεν εξέτασε την εφαρμογή των τεχνικών αμυντικής παραπλάνησης στις αρχιτεκτονικές τους και πως αυτές οι τεχνικές θα μπορούσαν να επηρεάσουν την ασφάλεια στα σύγχρονα δίκτυα. [1]

Εφαρμογές τεχνικών αμυντικής παραπλάνησης στα SDN

Απόκρυψη αναγνώρισης δικτύου με χρήση SDN εικονικών τοπολογιών

Στη συγκεκριμένη εργασία [34] ο S.Achleitner προσομοιώνει τοπολογίες δικτύου που βασίζονται στο SDN για να εξαπατήσουν επιτιθέμενους που στοχεύουν ένα δίκτυο. Το βασικό στοιχείο αυτής της τεχνικής παραπλάνησης είναι το Reconnaissance Deception System – RDS. Ένα SDN δίκτυο χρησιμοποιείται για την προσομοίωση τοπολογιών εικονικού δικτύου συμπεριλαμβανομένων των φυσικών συστατικών του. Το RDS αποτελείται από έναν Deception Server, έναν SDN Controller, ένα Honeypot server, έναν Delay Handler και μια γεννήτρια εικονικής τοπολογίας δικτύου. Το RDS είναι ένα σύστημα παραπλάνησης που στοχεύει στην υπεράσπιση ενός δικτύου από πιθανές επιθέσεις. Για το σκοπό αυτό, το RDS παραπληροφορεί τον εισβολέα για τις συσκευές και τα χαρακτηριστικά του δικτύου, υποδεικνύοντας μια εσφαλμένη ή εικονική τοπολογία του δικτύου. Για την ανάλυσή μας, θα εστιάσουμε στον τρόπο με τον οποίο θα μπορούσε ένα SDN να χρησιμοποιηθεί για να εξαπατήσει τους αντιπάλους.

Στην συγκεκριμένη τεχνική, ο SDN controller έχει σημαντικό ρόλο στην παραπλάνηση του αντιπάλου. Συγκεκριμένα, ο SDN controller είναι υπεύθυνος να δημιουργήσει δυναμικά κανόνες ροής και να αναλύσει στατιστικά τα στοιχεία ροής προκειμένου να εντοπίσει κάποια κακόβουλη δραστηριότητα. Οι κύριες λειτουργίες ενός SDN controller είναι οι παρακάτω:

- Η προώθηση των ARP requests στον Deception server: Με το χειρισμό όλων των πακέτων ARP μπορούμε να διασφαλίσουμε ότι οι υπολογιστές ενός δικτύου που πρέπει να παραμείνουν κρυφοί δεν θα τους ανακαλύψει κάποιος επιτιθέμενος.

- Αποστολή πακέτων με συγκεκριμένο TTL (Time To Live) στον Deception server: Μέσω αυτού του κανόνα μπορούμε να αλλοιώσουμε λειτουργίες όπως είναι το traceroute προκειμένου να αλλάξουμε τις πραγματικές διαδρομές ενός δικτύου.
- On the fly προσαρμογή του πεδίου TTL: Αυτός ο κανόνας στοχεύει στην προσαρμογή του πεδίου TTL των πακέτων απόκρισης (response packets) κατά τη διάρκεια της προώθησης των πακέτων μέσω των switch.
- Προώθηση των πακέτων σφαλμάτων ICMP στον Deception server: Τα πακέτα σφαλμάτων ICMP θα μπορούσαν να αποκαλύψουν σημαντικές πληροφορίες σχετικά με το δίκτυο. Μέσω αυτού του κανόνα, ο SDN controller προωθεί τα συγκεκριμένα πακέτα στον Deception server και προσαρμόζει κατάλληλα τις πληροφορίες που περιλαμβάνονται στα ένθετα πακέτα προτού φτάσουν στον προορισμό τους.
- Δρομολόγηση πακέτων DHCP: Αυτός ο κανόνας προωθεί τα πακέτα DHCP discover στον deception server.
- Δρομολόγηση πακέτων DNS: Ο SDN Controller δρομολογεί τα πακέτα DNS σύμφωνα με τους κανόνες ροής μεταξύ των κόμβων και του Deception server.
- Δρομολόγηση πακέτων από και προς τα Honeypots: Τα Honeypots κάνουν ένα δίκτυο μεγαλύτερο και συνεπώς πιο δύσκολο για έναν εισβολέα να το εξερευνήσει. Οι ροές που περνούν από τα Honeypots παρακολουθούνται από τον SDN controller και στη συνέχεια αναλύονται στατιστικά στοιχεία ώστε να εντοπίστουν οι εισβολείς.
- Dynamic Address Translation: Οι κεφαλίδες των πακέτων πρέπει να έχουν διαφορετική IP διεύθυνση προκειμένου να εξαπατήσουν τον επιτιθέμενο. Έτσι, ο SDN controller αλλάζει τις κεφαλίδες των πακέτων κατά τη διάρκεια της μετάδοσης σύμφωνα με το εικονικό δίκτυο που έχει δημιουργηθεί.
- Ετικέτα Πακέτων και Queuing: Αυτός ο κανόνας επιτρέπει στον SDN controller να προωθεί πακέτα σύμφωνα με συγκεκριμένες ουρές που είναι εγκατεστημένες σε κάποιο switch. Επιπλέον, ο SDN controller προσθέτει ετικέτες σε κάθε πακέτο, προκειμένου να αποφευχθεί συμφόρηση στο σύστημα.

Εν κατακλείδι, στη συγκεκριμένη εργασία χρησιμοποιείται το SDN για τη διαχείριση του πραγματικού και του εικονικού δικτύου με αποτελεσματικό και συγκεντρωτικό τρόπο. Η συγκεκριμένη τεχνική απαιτεί πολλούς κανόνες ροής ώστε να εξαπατήσει τον εισβολέα. Οι συγγραφείς επέλεξαν να χρησιμοποιήσουν SDN, δεδομένου ότι μέσω αυτής της υλοποίησης ήταν σε θέση να διατηρήσουν έναν κλιμακωτό αριθμό κανόνων ροής και να τους αναπτύξουν με ενεργό τρόπο. Επιπλέον, ένας κανόνα ροής αναπτύσσεται δυναμικά και εκπνέει αυτόματα μετά από μια σύντομη χρονική περίοδο.

Decoy Chain Development βασισμένη στα δίκτυα SDN

Ο Q.Zhao [35] στην τεχνική Decoy Chain Development (DCD) βασίζεται στο SDN και το NFV ως αμυντική τεχνική απέναντι στις επιθέσεις διείσδυσης (penetration attacks). Η decoy chain αποτελείται από μια σειρά από εικονικές μηχανές οι οποίες μπορούν να χρησιμοποιηθούν ως switches, middleboxes ή ως ηλεκτρονικοί υπολογιστές. Μια επίθεση σε κάποια από τις παραπάνω εικονικές μηχανές δεν επηρεάζει το πραγματικό δίκτυο μιας και αυτοί οι κόμβοι δεν αποτελούν μέρος του. Αυτή η τεχνική στοχεύει στην επιβράδυνση των κακόβουλων ενεργειών του αντιπάλου και μειώνει την πιθανότητα έκθεσης ευαίσθητων στόχων. Επιπλέον, ένα σημαντικό τμήμα αυτής της τεχνικής είναι ο SDN controller, ο οποίος παρακολουθεί την κατάσταση ασφαλείας ολόκληρου του δικτύου. Χρησιμοποιώντας αυτή τη λειτουργία του SDN το decoy chain θα μπορούσε να αναπτυχθεί με γνώση ολόκληρου του δικτύου και της κατάστασης ασφαλείας. Επιπλέον, οι decoy chains αναπτύσσονται στους εξυπηρετητές του επιπέδου δεδομένων προκειμένου να αλλάζουν την επιφάνεια επίθεσης του δικτύου. Συγκεκριμένα, υπάρχουν τρία επίπεδα στον σχεδιασμό του DCD: το επίπεδο πολιτικής (policy plane), το επίπεδο ελέγχου (control plane) και το επίπεδο δεδομένων (data plane). Ένας SDN controller χρησιμοποιείται για την υλοποίηση των λειτουργιών των επιπέδων πολιτικής και ελέγχου. Στο επίπεδο ελέγχου μια γεννήτρια τοπολογίας δικτύου θα μπορούσε να παρέχει πληροφορίες σχετικά με την τοπολογία του δικτύου και την ίδια στιγμή με την παρακολούθηση της κατάστασης ασφαλείας (security status monitor) θα μπορούσε να επιβλέπει το δίκτυο για ενδεχόμενη επίθεση. Από την άλλη πλευρά, στο επίπεδο πολιτικής, θα μπορούσαμε να ορίσουμε κάποιους ευαίσθητους στόχους προκειμένου να παρέχονται πληροφορίες για αυτούς σε όλο το δίκτυο. Έπειτα, ο SDN controller είναι σε θέση να ρυθμίζει το επίπεδο δεδομένων, να συλλέγει πληροφορίες ασφαλείας από το δίκτυο και να εντοπίζει οποιοδήποτε ζήτημα ασφαλείας του δικτύου. Επιπλέον, μια κρίσιμη λειτουργία του SDN controller στην υλοποίηση του DCD είναι ότι μπορεί να ελέγξει άμεσα τους servers στο επίπεδο δεδομένων και να αλλάξει την επιφάνεια επίθεσης του δικτύου αξιοποιώντας την υλοποίηση του decoy chain σε servers.

Συμπερασματικά, η συγκεκριμένη τεχνική έχει δοκιμαστεί με διάφορους αλγόριθμους που περιγράφουν διάφορες επιθέσεις διείσδυσης. Τα αποτελέσματα έδειξαν ότι η DCD θα μπορούσε να χρησιμοποιηθεί αποτελεσματικά για να μειώσει την πιθανότητα έκθεσης ευαίσθητων στόχων του δικτύου χρησιμοποιώντας κάποιον SDN controller.

HoneyProxy

Ο S.Kyung [37] σχεδίασε ένα δίκτυο από honeypots, το honeynet, το οποίο είναι βασισμένο στο SDN με σκοπό την παρακολούθηση της κίνησης του δικτύου με τη βοήθεια του SDN controller. Τα υπάρχοντα honeynets δεν επαρκούν για τα σύγχρονα δίκτυα υπολογιστών, καθώς έχουν περιορισμένες λειτουργίες στον έλεγχο και στη συλλογή δεδομένων. Για το σκοπό αυτό, έχει προταθεί ένα προηγμένο honeynet με την ονομασία HoneyProxy. Το HoneyProxy είναι σε θέση να βελτιώσει τις δυνατότητες συλλογής δεδομένων αξιοποιώντας τον SDN controller. Ως εκ τούτου, το honeynet παρέχει μεγαλύτερη ευελιξία όσον αφορά τη διαχείριση της πρόσβασης στο δίκτυο. Με το συνδυασμό αυτών των δυο στοιχείων, είναι σε θέση να ανιχνεύσει τυχόν ανωμαλίες στο δίκτυο.

Η παραπάνω υλοποίηση αντιμετωπίζει τα παρακάτω θέματα σε ένα δίκτυο:

- Figerprinting attacks targeting honeypots: Οι υπάρχουσες αρχιτεκτονικές honeypot είναι εύκολα ανιχνεύσιμες από τους εισβολείς. Αυτός ο περιορισμός επηρεάζει την αποτελεσματικότητα της συλλογής στοιχείων σχετικά με τη συμπεριφορά του εισβολέα.
- Internal malware propagation in honeynet: Ο εισβολέας είναι σε θέση να μολύνουν ένα honeynet, απλώς μολύνοντας με κακόβουλο λογισμικό μόνο ένα honeypot στο δίκτυο.
- Lack of honeypot transition: Σε ένα honeynet υπάρχουν δυο τύποι honeypots, τα honeypots χαμηλής αλληλεπίδρασης (LIH) και τα honeypots υψηλής αλληλεπίδρασης (HIH). Η πρώτη κατηγορία λειτουργεί στο πρώτο στάδιο ανίχνευσης επίθεσης και η δεύτερη κατηγορία εφαρμόζει πραγματικές υπηρεσίες όπως είναι το SSH και το HTTP και απαιτεί υψηλό κόστος συντήρησης και πολύπλοκη διαμόρφωση (configuration). Έτσι, τα honeynets εξαρτώνται απόλυτα από την ικανότητα κάθε honeypot δεδομένου ότι δεν υπάρχει άμεση επικοινωνία μεταξύ LIH και HIH.

Εστιάζοντας στην εφαρμογή του SDN στην συγκεκριμένη τεχνική, έχει χρησιμοποιηθεί ένας SDN Controller για τη διαμόρφωση του δικτύου και για την επιβολή κανόνων ασφαλείας σε όλο το δίκτυο. Επιπλέον, με τη χρήση του SDN, μπορούν να εντοπιστούν ύποπτα πακέτα μέσα στο δίκτυο. Η εφαρμογή του SDN αποτελείται από τα παρακάτω:

- Flow Programming Module: Είναι υπεύθυνο να ειδοποιήσει τον Controller και να προσθέσει SDN κανόνες σύμφωνα με την ανάλυση της κίνησης του δικτύου.
- Behavior Tracker: Ενημερώνει τον proxy να αλλάξει κατάσταση λειτουργίας όταν είναι απαραίτητο σύμφωνα με συγκεκριμένα κριτήρια.

Επιπλέον, ένα honeyproxy χρησιμοποιώντας έναν SDN Controller παρακολουθεί τις ροές δεδομένων στο δίκτυο και πραγματοποιεί τις κατάλληλες παρεμβάσεις στους proxy servers όταν ένα honeypot μολυνθεί. Στην εφαρμογή του honeyproxy, ο ρόλος του SDN Controller είναι

κρίσιμος δεδομένου ότι μέσω αυτής της τεχνολογίας επιτυγχάνεται η διαμόρφωση και η διαχείριση του δικτύου. Παρόλο που τα honeypots αποτελούν μια από τις παραδοσιακές τεχνικές άμυνας, δεν υπάρχουν πολλές εφαρμογές στην τεχνολογία SDN.

HoneyMix

Η συγκεκριμένη τεχνική [33] αποτελείται από ένα "έξυπνο" honeynet το οποίο βασίζεται στο SND με σκοπό να προσελκύσει επιτιθέμενους ώστε να μάθουμε την τακτική και την συμπεριφορά τους. Με την αξιοποίηση της τεχνολογίας SDN, τα Honeynets είναι σε θέση να αποφύγουν τις επιθέσεις fingerprinting οι οποίες είναι πολύ συνηθισμένες σε τέτοιους αμυντικούς μηχανισμούς. Επιπλέον, η τεχνική HoneyMix χρησιμοποιεί την τεχνολογία SDN σε διαφορετικά επίπεδα της αρχιτεκτονικής της ώστε να επιτρέπει πιο αποτελεσματικό και αποδοτικό έλεγχο δεδομένων. Συγκεκριμένα, τα μέρη του HoneyMic που εφαρμόζουν την τεχνολογία SDN είναι τα παρακάτω:

- Forwarding Decision Engine (FDE): Αξιοποιεί τα SDN switches προκειμένου να περάσει όλες τις κακόβουλες δραστηριότητες στα honeypots.
- Connection Selection Engine (CSE): Συνδέει τους κόμβους του εισβολέα με τα honeypots. Το HoneyMix διατηρεί αυτές τις συνδέσεις για κάθε πιθανή επίθεση χρησιμοποιώντας ένα SDN switch ανάμεσα στον εισβολέα και στο honeypot.
- Behavior Learner: Με τη χρήση ενός SDN switch το HoneyMix αναλύει τη δραστηριότητα του εισβολέα προκειμένου να κατανοήσει την τακτική και την συμπεριφορά του.
- SDN switches: Το κύριο καθήκον των SDN switches στο HoneyMix είναι να χειρίζονται τη ροή των δεδομένων ανάμεσα στον controller και στα honeypots. Συγκεκριμένα, ένα SDN switch είναι σε θέση να απομονώνει ένα μολυσμένο honeypot και δημιουργήσει μια νέα σύνδεση με ένα άλλο honeypot του δικτύου.

Η δυνατότητα προγραμματισμού στο SDN προσφέρει δυναμική διαμόρφωση των κανόνων του δικτύου ανάλογα με την περίσταση. Επιπλέον, το SDN switch επιτρέπει την άμεση σύνδεση μεταξύ των honeypots και ως εκ τούτου, αυξάνεται η προστασία από επιθέσεις fingerprinting στα honeypots. Δηλαδή, εάν ένα honeypot μολυνθεί από έναν κακόβουλο χρήστη, το SDN είναι σε θέση να απομονώσει αυτή τη σύνδεση και παράλληλα να δημιουργήσει μια άλλη σύνδεση σε ένα μη μολυσμένο honeypot. Συμπερασματικά, ο σκοπός του HoneyMix είναι να παρέχει προστασία από επιθέσεις fingerprinting και να ανταποκριθεί σε περιστατικά ασφαλείας χρησιμοποιώντας την τεχνολογία SDN.

Moving Target Defense βασισμένη στα δίκτυα SDN

Η τεχνική Moving Target Defense είναι μια ακόμη τεχνική αμυντικής παραπλάνησης με πολλές υλοποιήσεις στα παραδοσιακά δίκτυα υπολογιστών. Σε αυτή την τεχνική [39], χρησιμοποιείται ένας SDN Controller προκειμένου να αναδιαμορφώνει δυναμικά το δίκτυο και να δυσκολέψει έναν εισβολέα να κατανοήσει την τοπολογία του δικτύου. Συγκεκριμένα, ο στόχος του MTD είναι να αλλάζει δυναμικά την επιφάνεια επίθεσης του δικτύου. Το SDN θεωρείται ως η κατάλληλη τεχνολογία αφού είναι σε θέση να αναδιαμορφώνει συνεχώς την τοπολογία του δικτύου. Σε αυτή την τεχνική ο αριθμός της πόρτας κάθε υπηρεσίας (service port) ή η διεύθυνση IP μιας εικονικής μηχανής αλλάζει συνεχώς. Σε αυτή τεχνική έχει χρησιμοποιηθεί ο Opendaylight SDN Controller ο οποίος ελέγχει τις εικονικές μηχανές μέσω του OpenvSwitch. Επίσης, ο SDN controller είναι υπεύθυνος για το επίπεδο ελέγχου. Επιπλέον, υπάρχει άμεση αλληλεπίδραση με τον Log Analyzer προκειμένου λάβει πληροφορίες για πιθανά θέματα ασφαλείας. Ο SDN controller ενημερώνει τους κανόνες ροής και την τοπολογία του δικτύου και επικοινωνεί με Vulnerability Scanner (Nessus) προκειμένου να ελέγξει για νέες ευπάθειες στο δίκτυο.

Moving Target Defense βασισμένη στα δίκτυα SDN

Μια ακόμη εργασία στην MTD έχει γίνει από τον P.kampanakis [40], όπου εξετάστηκε η εφαρμογή της τεχνολογίας SDN σε κάποιες τεχνικές MTD με βάση το δίκτυο. Η χρήση του SDN σε τεχνικές MTD έχει ως στόχο να υπονομεύσει τις ενέργειες ενός εισβολέα. Για παράδειγμα, εάν υπάρχει μεγάλη κίνηση σε ένα κόμβο του δικτύου, το SDN θα μπορούσε να δημιουργήσει ποικίλες απαντήσεις που θα μπερδέψουν ακόμη περισσότερο τον εισβολέα, καθώς η κακόβουλη δραστηριότητα δεν θα διακοπεί αμέσως. Συνεπώς, το SDN θα μπορούσε να βελτιώσει το μηχανισμό άμυνας απέναντι στο port scanning είτε αυτό αφορά TCP, UDP ή ICMP. Παρακάτω ακολουθεί ο αλγόριθμος ο οποίος περιγράφει τα βήματα που θα μπορούσαν να υλοποιηθούν στο SDN ώστε να αλλοιωθούν τα αποτελέσματα ενός TCP port scanning.

```
Require: Probabilities Pr_SA < Pr_A < Pr_PA < Pr_R < 1
  hash table action_buffer ← NULL
  while (new TCP packet p is received) do
    if (p is illegitimate traffic) then
      if (p.dest_port not in action_buffer) then
        r ← random real number ∈ [0, 1]
        store r in action_buffer
      else
        r ← as in action_buffer
      end if
      switch (r)
       case r < Pr_SA:
        respond with TCP SYN-ACK
       case r < Pr_A:
        respond with TCP ACK
       case r < Pr_PA:
        respond with TCP PUSH-ACK with random payload
       case r < Pr_R:
        respond with TCP RST packet
       default:
        drop silently
      end switch
    end if
  end while
```

Figure 13: Αλγόριθμος αλλοίωσης ενός TCP port scanning.

Ο συγκεκριμένος αλγόριθμος θα μπορούσε, με κάποιες τροποποιήσεις, να χρησιμοποιηθεί για την προστασία του δικτύου από DoS επιθέσεις. Επιπλέον, μέσω της προτεινόμενης υλοποίησης, το SDN θα μπορούσε να ανοίξει ψευδείς πόρτες οι οποίες θα σχετίζονται με μια πραγματική υπηρεσία στο δίκτυο και έτσι να μπερδέψουν τον εισβολέα. Μια ακόμη σημαντική λειτουργία του SDN είναι ότι μπορεί να κρύψει τις πραγματικές εκδόσεις των υπηρεσιών (service versions), προκειμένου να αποφευχθεί η διαρροή πληροφοριών του δικτύου, όπως για παράδειγμα είναι η έκδοση του λειτουργικού συστήματος των υπολογιστών. Η επίθεση που στοχεύει στην αναγνώριση του λειτουργικού συστήματος ονομάζεται OS fingerprinting και η τεχνολογία SDN είναι σε θέση να προστατεύσει το δίκτυο από μια τέτοια επίθεση. Έτσι, ο SDN Controller παράγει TCP responses και payloads που αναφέρονται σε υπάρχοντα προφίλ λειτουργικών συστημάτων. Επομένως, ένας εισβολέας δεν είναι σε θέση να προσδιορίσει το πραγματικό λειτουργικό σύστημα και να εκτελέσει τις αντίστοιχες επιθέσεις.

Εν κατακλείδι, μέσω αυτής της τεχνικής παρουσιάζονται οι πιθανές εφαρμογές του SDN οι οποίες παρέχουν παραμορφώσεις (obfuscations) στο δίκτυο με σκοπό να αυξηθεί το κόστος μιας επίθεσης και ένας εισβολέας να χρειάζεται να δαπανήσει περισσότερους πόρους για να προσδιορίσει την τοπολογία του δικτύου. Η εφαρμογή τεχνικών MTD είναι ευκολότερη όταν χρησιμοποιείται η τεχνολογία SDN για την παρακολούθηση και τον έλεγχο του δικτύου.

DDoS Defense χρησιμοποιώντας Moving Target Defense και δίκτυα SDN

Στη συγκεκριμένη εργασία, από τον J.Steinberger [41], έχουμε συνδυασμό των MTD και SDN προκειμένου να μειωθούν οι επιδράσεις μιας κυβερνοεπίθεσης μεγάλης κλίμακας. Η υλοποίηση ενός τέτοιου συνδυασμού είναι χρήσιμη στους ISPs, δεδομένου ότι κυβεροεπιθέσεις μεγάλης κλίμακας συνήθως στοχεύουν τις βασικές υποδομές του δικτύου. Η στατική διαμόρφωση των παραδοσιακών δικτύων υπολογιστών τις καθιστά ευάλωτες σε επιθέσεις DDoS, καθώς είναι εύκολο για τους επιτιθέμενους να αναγνωρίσουν το δίκτυο και να επιλέξουν την πιο αποτελεσματική επίθεση για να προκαλέσουν ζημιά. Για να ξεπεραστεί το πρόβλημα της στατικής διαμόρφωσης του δικτύου, οι δημιουργοί της συγκεκριμένης τεχνικής προτείνουν την εφαρμογή των μηχανισμών SDN και MTD.

Η χρήση του SDN προσφέρει δυνατότητα επέκτασης, αφού ένα απλό switch είναι σε θέση να χειριστεί πολυάριθμα IP prefixes και κανόνες αντιστοίχισης (matching rules) μέσω του δικτύου. Επίσης, συνδυάζοντας τις τεχνικές SDN και MTD δεν είναι απαραίτητη η εγκατάσταση πρόσθετου hardware στις υποδομές των ISPs. Συσκευές όπως είναι τα middleboxes μπορούν να ελεγχθούν εξ αποστάσεως ή να κατασκευάζονται τεχνητά από το SDN.

Επιπλέον, η εφαρμογή του SDN σε αυτό το μηχανισμό άμυνας DDoS επιθέσεων παρακολουθεί τη ροή μέσω του δικτύου, καθιερώνει την ανταλλαγή συμβάντων ασφαλείας και συμβάλει στη διαδικασία συνεργασίας μεταξύ των ISPs και των αξιόπιστων μερών (trusted parties). Οι συγγραφείς αυτής της εργασίας επέλεξαν το λειτουργικό σύστημα ONOS , για να εφαρμόσουν την τεχνική MTD. Πλεονέκτημα του ONOS είναι ότι εξασφαλίζει επεκτασιμότητα σε δίκτυα υψηλών ταχυτήτων. Σε αυτό το μηχανισμό άμυνας χρησιμοποιούνται δυο τεχνικές MTD: (α) MTD σε επίπεδο δικτύου, η οποία βασίζεται σε BGP διαδρομές (routes) και πολλαπλούς routers και (β) MTD σε επίπεδο κεντρικού υπολογιστή, η οποία πραγματοποιεί IP Hopping για να δημιουργήσει ένα honeypot.

Συμπερασματικά, αυτή η τεχνική επικεντρώνεται στην χρήση του ONOS SDN OS για την ανάπτυξη ενός αμυντικού μηχανισμού για επιθέσεις DDoS. Οι συγγραφείς αξιοποιούν τη δυνατότητα προγραμματισμού και επέκτασης στα SDN προκειμένου να επιτευχθεί καλύτερη επικοινωνία και συνεργασία μεταξύ των παρόχων υπηρεσιών διαδικτύου και τρίτων. Η εφαρμογή της τεχνικής MTD χρησιμοποιώντας την τεχνολογία SDN είναι αποτελεσματική για την πρόληψη ή την άμβλυνση μιας επίθεσης DDoS. Αν και οι τεχνικές MTD είναι αποτελεσματικές σε δίκτυα μεγάλης κλίμακας, η εφαρμογή τους σε μικρότερα δίκτυα μπορεί να έχει σημαντικές διαφορές στα αποτελέσματα.

Evolutionary Computation για Moving Target Defense σε δίκτυα SDN

Σε αυτή την εργασία ο A.Makanju [42] προτείνει μια τεχνική Evolutionary Computation (EC) για Moving Target Defence σε συνδυασμό με την τεχνολογία SDN. Συγκεκριμένα, έχουν σχεδιαστεί αλγόριθμοι EC για την αποτελεσματική αναζήτηση μεγάλων χώρων για βέλτιστες λύσεις. Έτσι, η EC αποτελεί την ιδανική λύση για την εφαρμογή τεχνικών MTD σε μεγάλα δίκτυα υπολογιστών. Σύμφωνα με τους συγγραφείς, η τεχνική MTD θα είναι σε θέση να αναπτύξει μια νέα διαμόρφωση για το δίκτυο, λαμβάνοντας υπόψη την κατάσταση του δικτύου και τις ειδοποιήσεις εισβολής (intrusion alerts). Επιπλέον, κάθε επιλεγμένη διαμόρφωση πρέπει να εφαρμόζεται με τις αντίστοιχες πολιτικές, όπως η Service Level Agreement.

Η συγκεκριμένη εφαρμογή της τεχνολογίας SDN, εγγυάται ότι η τεχνική MTD είναι σε θέση να εντοπίσει και να αναπτύξει ένα ευρύ φάσμα διαμορφώσεων δικτύου. Η MTD αξιοποιεί την τεχνολογία SDN προκειμένου να αποτρέψει επιθέσεις, καθώς δεν χρειάζεται να διαμορφώνει το δίκτυο συχνά.

Minimal Moving Target Defense χρησιμοποιώντας δίκτυα SDN

Σε αυτή την εργασία ο S.Debroy [43] προτείνει την εφαρμογή της τεχνικής MTD σε μια υποδομή cloud βασισμένη σε SDN. Αυτή η προσέγγιση στοχεύει στην ελαχιστοποίηση του κόστους διαχείρισης χρησιμοποιώντας MTD σε ετερογενείς εικονικές μηχανές. Για το σκοπό αυτό, προτείνεται μια κατάλληλη τοποθεσία VM χρησιμοποιώντας έναν SDN controller ο οποίος κατευθύνει τα Open Flow switches στην υποδομή του cloud.

Η προτεινόμενη αρχιτεκτονική SDN περιγράφει την τεχνική MTD χρησιμοποιώντας VMs. Ο στόχος της τεχνικής MTD είναι να μετακινεί τους κόμβους που αποτελούν στόχο σε ένα VM. Όμως η ευθύνη για τη μετακίνηση είναι διπλή αφού αρχικά διευκολύνει τη διαχείριση της συχνότητας μετακίνησης του VM προκειμένου να αποφευχθούν οι επιθέσεις DoS και στη συνέχεια η ευθύνη για τη θέση μετακίνησης επιτρέπει τη διαδικασία επιλογής νέου VM όταν αυτό είναι απαραίτητο.

Επιπλέον, η μονάδα ελέγχου ξεκινά τη διαδικασία μετάβασης μέσω της μονάδας εκκίνησης μετάβασης και στη συνέχεια ανακατευθύνονται σε VM χρησιμοποιώντας OpenFlow switches. Η μονάδα ανίχνευσης εισβολής εξασφαλίζει τη διαδικασία μετακίνησης ανιχνεύοντας επιθέσεις DoS. Σε αυτή την εργασία, έχει προταθεί ένα σχέδιο μετακίνησης VM με την αξιοποίηση της τεχνολογίας SDN. Μέσω του SDN Controller, είναι εφικτή η αποτελεσματική αλλαγή των κόμβων του δικτύου, προκειμένου να αποφευχθούν επιθέσεις όπως είναι η DoS.

OpenFlow Random Host Mutation χρησιμοποιώντας δίκτυα SDN

Στη συγκεκριμένη MTD τεχνική, ο J.H.Jafarian [44], χρησιμοποιεί έναν SDN Controller για να αλλάζει δυναμικά τις IP διευθύνσεις κάθε υπολογιστή. Η τεχνική OpenFlow Random Host Mutation (OF-RHM) αποδίδει μια τυχαία εικονική διεύθυνση IP η οποία μεταφράζεται από / προς την πραγματική διεύθυνση IP του υπολογιστή, προκειμένου να ξεπεραστεί η στατική διαμόρφωση των παραδοσιακών δικτύων υπολογιστών. Σκοπός της τεχνικής αυτής είναι η προστασία της τοπολογίας του δικτύου από τη κρυφή σάρωση (stealthy scanning), τη διάδοση worms και άλλες επιθέσεις scanning.

Ο SDN controller είναι υπεύθυνος για τις ακόλουθες εργασίες:

- Συντονισμένη αλλαγή των IP διευθύνσεων,
- Ρύθμιση των νέων IPs χρησιμοποιώντας SMT,
- Διαχειρίζεται τις ενεργές συνδέσεις του δικτύου,
- Διαχειρίζεται τις ενημερώσεις DNS.

Σε αυτή την εργασία χρησιμοποιείται ένας NOX Controller για τη διαχείριση ολόκληρου του δικτύου. Ο NOX controller διαχειρίζεται την αλλαγή των διευθύνσεων IP, την εγκατάσταση ροής στα switches και τις αποκρίσης DNS. Επιπλέον, έχει σχεδιαστεί ένας αλγόριθμος για τον NOX Controller, ο οποίος παρουσιάζεται στην παρακάτω εικόνα, σύμφωνα με τον οποίο τα OpenFlow switches ενσωματώνουν τα πακέτα που χαρακτηρίζονται ως unmatched και τα στέλνουν στον SDN controller. Έπειτα, ο SDN Controller προσδιορίζει τον τύπο της σύνδεσης και εγκαθιστά τις απαραίτητες ροές στα switches.

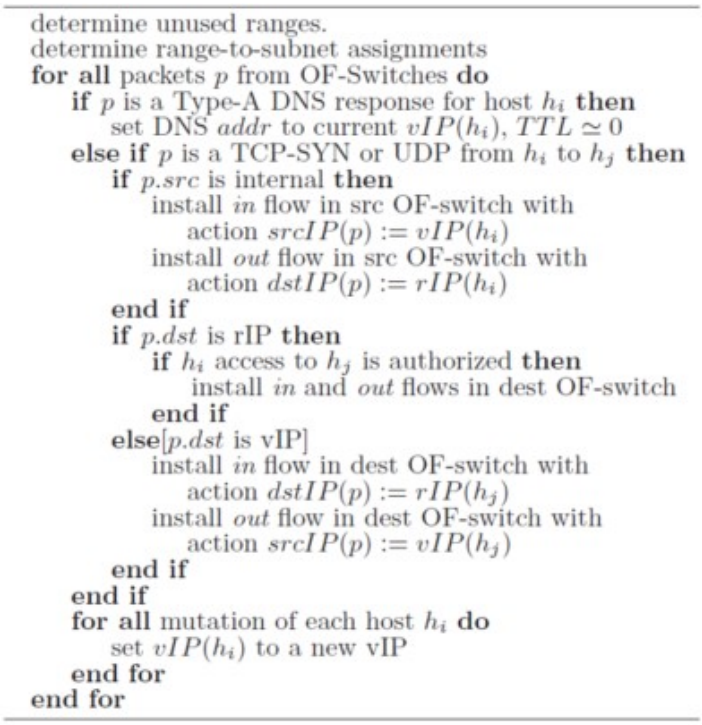


Figure 14: Αλγόριθμος IP Address mutation

Ο SDN controller επικοινωνεί με του υπολογιστές χρησιμοποιώντας είτε το όνομα του υπολογιστή είτε την πραγματική IP διεύθυνση (rIP) του υπολογιστή. Αυτή η αμυντική τεχνική παραπλάνησης χρησιμοποιώντας την τεχνολογία SDN αυξάνει τον ρυθμό της απρόβλεπτης μετάλλαξης, καθώς η μετάλλαξη μπορεί να επιτευχθεί σε μεγαλύτερο εύρος από τις υπάρχουσες υλοποιήσεις MTD. Λαμβάνοντας υπόψη τα αποτελέσματα, αυτή η μελέτη υποστηρίζει ότι μέσω του OF-RHM η συλλογή πληροφοριών από scanners μπορεί να μειωθεί έως και 99% και ταυτόχρονα οι υπολογιστές να προστατευθούν από zero-day worms έως 90%. Ένας περιορισμός αυτής της εργασίας είναι ότι λαμβάνεται υπόψη μόνο η προστασία από τη σάρωση IP και όχι η σάρωση DNS.

Αντιμετώπιση επιθέσεων Crossfire χρησιμοποιώντας SDN-based Moving Target Defense

Στην εργασία του ο A.Aydeger [46] προτείνει ένα μηχανισμό άμυνας για επιθέσεις Crossfire DDoS. Συγκεκριμένα, ο συγγραφέας αξιοποιώντας την τεχνολογία SDN και την τεχνική MTD στοχεύει στη δυναμική αναδιαμόρφωση του δικτύου προκειμένου να παραπλανήσει τους επιτιθέμενους. Ο SDN controller επιτρέπει τη διαχείριση της ροής της κίνησης του δικτύου και είναι σε θέση να προστατεύσει το δίκτυο από proactive και reactive επιθέσεις. Ωστόσο, αυτή η εργασία επικεντρώνεται ιδιαίτερα στις crossfire επιθέσεις. Τέτοιες επιθέσεις αποσκοπούν στην επιβάρυνση του δικτύου αξιοποιώντας ευπάθειες κρίσιμων συνδέσεων.

Αυτός ο μηχανισμός άμυνας αποτελείται από τα ακόλουθα τέσσερα αλληλοσυνδεόμενα στοιχεία του SDN.

- **ICMP monitoring**: Μέσω του πρωτοκόλλου OpenFlow, άλλα τα πακέτα ICMP αποστέλλονται στον SDN controller για έλεγχο.
- **Traceroute profiling**: Αυτή η υπομονάδα δημιουργεί μια traceroute profile βάση δεδομένων η οποία περιέχει την IP διεύθυνση προέλευσης, την IP διεύθυνση προορισμού, χρονική σήμανση (timestamp), το πλησιέστερο SDN switch στη διεύθυνση προορισμού και όλα τα ενδιάμεσα switches. Επιπλέον, μέσα από αυτό το module εντοπίζονται πιθανοί σύνδεσμοι - στόχοι στους οποίους μπορούν να επιτεθούν.
- **Route mutation**: Αυτή η υπομονάδα του SDN controller είναι υπεύθυνη για τη μετάλλαξη μιας σύνδεσης μεταξύ πηγής και προορισμού όταν αναγνωρίζεται μια κακόβουλη δραστηριότητα. Επιπλέον, ο SDN controller εξετάζοντας την υπομονάδα traceroute profiling θα αναλύσει την εγκυρότητα της νέας σύνδεσης.
- **Congestion-link monitoring**: Ο SDN controller προσδιορίζει όλες τις πιθανές διαδρομές από ένα ύποπτο host έως τον προορισμό.

Στον προτεινόμενο μηχανισμό χρησιμοποιείται ο SDN Controller FloodLight. Επιπλέον, ο περιγραφόμενος αμυντικός μηχανισμός παραπλάνησης έλαβε υπόψη μόνο πακέτα ICMP και δεν εξέτασε πακέτα TCP / UDP.

Σύγκριση αμυντικών τεχνικών παραπλάνησης

Μέχρι τώρα αναλύσαμε τις υφιστάμενες αμυντικές τεχνικές παραπλάνησης στις τεχνολογίες SDN. Ειδικότερα, αναλύσαμε έντεκα διαφορετικές προσεγγίσεις που αξιοποιούν τις δυνατότητες της τεχνολογίας SDN. Όπως αναφέραμε παραπάνω, σύμφωνα με τον F.Cohen [15] οι αμυντικές τεχνικές παραπλάνησης θα πρέπει να πληρούν πέντε ιδιότητες για την αποτελεσματική υπεράσπιση του δικτύου. Ο παρακάτω πίνακας παρουσιάζει τις τεχνικές που έχουν αναλυθεί και τις ιδιότητες που ικανοποιεί κάθε μία.

Reference	Ιδιότητα 1	Ιδιότητα 2	Ιδιότητα 3	Ιδιότητα 4	Ιδιότητα 5
[34]	✓	✓			✓
[35]	✓	✓	✓		
[37]		✓		✓	
[33]	✓	✓		✓	✓
[39]	✓		✓	✓	✓
[40]			✓	✓	✓
[41]	✓	✓	✓		
[42]	✓		✓	✓	✓
[43]	✓	✓		✓	✓

[44]	✓			✓	✓
[46]	✓	✓		✓	✓
	9	8	5	9	9

Table 5: Σύγκριση αμυντικών τεχνικών παραπλάνησης

Όπως παρατηρούμε, πέντε από τις έντεκα υλοποιήσεις έχουν λάβει υπόψη τέσσερις ιδιότητες, ενώ έξι ικανοποιούν τις τρείς από τις πέντε ιδιότητες. Μπορούμε να συμπεράνουμε ότι οι τεχνικές MTD επικεντρώνονται κυρίως στην αύξηση του φόρτου εργασίας και της αβεβαιότητας του εισβολέα. Από την άλλη πλευρά, τεχνικές όπως είναι τα honeypots ή το honeyproxy αποσκοπούν στην αναγνώριση της επίθεσης προτού οι εισβολείς πετύχουν τον σκοπό τους. Επιπλέον, μόνο πέντε από τις αναλυθείσες προσεγγίσεις αποσκοπούν στην εξάντληση των πόρων του εισβολέα. Παρόλο που η τεχνολογία SDN είναι σε θέση να ανταποκριθεί στις προαναφερθείσες ιδιότητες αμυντικής παραπλάνησης, δεν υπάρχουν υλοποιήσεις που να τις εξετάζουν όλες.

Πρακτικό Μέρος

Η στατική φύση των δικτύων υπολογιστών επιτρέπει στους εισβολείς να εκτελούν αναγνωρίσεις στα δίκτυα και να εντοπίζουν ευπάθειες τις οποίες μπορούν να αξιοποιήσουν στη συνέχεια μέσω προηγμένων επιθέσεων. Οι επιτιθέμενοι εξετάζουν τα δίκτυα προσπαθώντας να αναγνωρίσουν κεντρικούς υπολογιστές, ανοικτές πόρτες και να χαρτογραφήσουν την τοπολογία του δικτύου με σκοπό να βρουν γνωστές ή άγνωστες (zero-day) ευπάθειες, με βάση τις οποίες θα συνεχίσουν την επίθεση τους.

Σε αυτή την εργασία, θα προσπαθήσουμε να παραπλανήσουμε τέτοιες τεχνικές κακόβουλων αναγνωρίσεων παρουσιάζοντας μια εικονική τοπολογία του δικτύου η οποία κρύβει το πραγματικό δίκτυο και πιθανές ευπάθειες τις οποίες μπορούν να εκμεταλλευτούν οι εισβολείς. Η παρουσίαση ενός εικονικού δικτύου ακυρώνει το σύνολο των πληροφοριών που συλλέγει ένας εισβολέας από την αναγνώριση ενός δικτύου και επιτυγχάνει καθυστέρηση στο ρυθμό αναγνώρισης ευάλωτων κεντρικών υπολογιστών. Με αυτή τη διαδικασία κερδίζουμε επιπλέον χρόνο που μπορεί να χρησιμοποιηθεί για τον εντοπισμό εισβολέα και την απομόνωσή του από το δίκτυο.

Τεχνικές όπως είναι η κοινωνική μηχανική, οι zero-day ευπάθειες, οι επιθέσεις client-side μπορούν να προκαλέσουν σημαντικές ζημιές και είναι δύσκολο να εντοπιστούν. Στο μοντέλο απειλής (threat model) αυτής της εργασίας θεωρούμε ότι οι εισβολείς βρίσκονται μέσα στο δίκτυο και έχουν μολύνει έναν υπολογιστή και το προτεινόμενο σύστημα αμυντικής παραπλάνησης αντιμετωπίζει δραστηριότητες αναγνώρισης στο δίκτυο. Ένα σημαντικό κομμάτι του μηχανισμού παραπλάνησης είναι η σύνθεση του εικονικού δικτύου, η τοποθέτηση honeypots και η προσομοίωση συνεκτικών χαρακτηριστικών δικτύου στις εικονικές τοπολογίες για να καθυστερήσει τους εισβολείς από τον εντοπισμό των πραγματικών και ευάλωτων υπολογιστών στο δίκτυο. Το σύστημά μας, εκχωρεί σε κάθε κεντρικό υπολογιστή μια διαφορετική προβολή του εικονικού δικτύου έτσι ώστε ο μηχανισμός παραπλάνησης να είναι ανεξάρτητος από την πηγή των κακόβουλων ανιχνεύσεων στο δίκτυο, η οποία υποθέτουμε ότι αρχικά δεν είναι γνωστή.

Ο μηχανισμός παραπλάνησης αποτελείται από τέσσερα βασικά στοιχεία, έναν ελεγκτή SDN ο οποίος είναι υπεύθυνος για τη δυναμική δημιουργία και διαχείριση των κανόνων ροής ώστε να κατευθύνει και να ελέγχει την κυκλοφορία του δικτύου, έναν Deception Server ο οποίος είναι υπεύθυνος για τον χειρισμό των πακέτων του δικτύου και για την προσομοίωση συγκεκριμένων πόρων του εικονικού δικτύου, μια γεννήτρια εικονικού δικτύου που περιέχει μια περιγραφή των στοιχείων του εικονικού δικτύου και της συνδεσιμότητας του, καθώς και τον Honeypot Server σε μια εικονική τοπολογία.

Όταν φτάσει ένα πακέτο σε ένα SDN switch, ο ελεγκτής εφαρμόζει έναν κανόνα ροής σύμφωνα με την εικονική τοπολογία του δικτύου, σύμφωνα με την οποία είτε το προωθεί στον Deception server, αν πρόκειται για ARP, ICMP, UDP, είτε το στέλνει στον κεντρικό υπολογιστή προορισμού, αν πρόκειται για TCP μήνυμα. Εάν ένα πακέτο αποστέλλεται στον Deception server, δημιουργείται ένα πακέτο απάντησης σύμφωνα με την εικονική τοπολογία και αποστέλλεται πίσω στην πηγή. Αν ένα πακέτο έχει σαν προορισμό κάποιο honeypot, τότε η πηγή του πακέτου χαρακτηρίζεται ως εισβολέας. Επίσης, όταν από μια πηγή αποσταλούν πολλά SYN πακέτα και προορισμός είναι πολλές πόρτες ενός πραγματικού host τότε η πηγή χαρακτηρίζεται και πάλι ως εισβολέας.

Το παραπάνω σύστημα έχει αναπτυχθεί με χρήση της γλώσσας προγραμματισμού Python. Ο ελεγκτής POX έχει χρησιμοποιηθεί για την υλοποίηση του ελεγκτή SDN και το Scapy framework

για την υλοποίηση του Deception server. Ο παραπάνω μηχανισμός αμυντικής παραπλάνησης έχει αναπτυχθεί στο Mininet, ο οποίος είναι ένας σύγχρονος εξομοιωτής δικτύων SDN. Το πρωτόκολλο επικοινωνίας μεταξύ του ελεγκτή SDN και του switch SDN είναι το OpenFlow 1.3.

Η υλοποίηση του πρακτικού μέρους της εργασίας αποτελείται από τέσσερις βασικές οντότητες:

- τον ελεγκτή POX
- μια τοπολογία δικτύου στο mininet
- έναν χειριστή πακέτων (PacketHandler)
- έναν Honeypot Server

Ο ελεγκτής POX

Ο κώδικας του ελεγκτή POX βρίσκεται στο directory: /home/mininet/pox/pox/ibc. Το συγκεκριμένο directory περιέχει τα παρακάτω python scripts:

```
mininet@mininet-vm:~/pox/pox/ibc$ pwd
/home/mininet/pox/pox/ibc
mininet@mininet-vm:~/pox/pox/ibc$ ls -ltr
total 84
-rw-r--r-- 1 root root     0 Ιουν  7 14:15 __init__.py
-rw-r--r-- 1 root root  1749 Ιουν  8 16:47 networkView.txt
-rw-r--r-- 1 root root   124 Ιουν 18 21:58 __init__.pyc
-rw-r--r-- 1 root root 14274 Ιουν 22 22:08 networkReader.py
-rw-r--r-- 1 root root  7524 Ιουν 22 22:12 networkReader.pyc
-rw-r--r-- 1 root root 20645 Ιουλ  3 19:49 iController.py
-rw-r--r-- 1 root root  9045 Ιουλ  3 19:50 iController.pyc
-rw-r--r-- 1 root root  7076 Ιουλ  3 20:00 createNetworkTopology.py
-rw-r--r-- 1 root root  6192 Ιουλ  3 20:00 createNetworkTopology.pyc
mininet@mininet-vm:~/pox/pox/ibc$
```

Figure 15: Αρχεία python από τα οποία αποτελείται ο ελεγκτής.

όπου:

- το networkView.txt περιέχει την τοπολογία του δικτύου
- το networkReader.py διαβάζει την τοπολογία του δικτύου από το networkView.txt
- το iController.py περιέχει τον κώδικα του ελεγκτή
- το createNetworkTopology.py δημιουργεί το αρχείο networkView.txt

Ένα παράδειγμα της εικονικής τοπολογίας που περιέχει το αρχείο networkView.txt είναι το παρακάτω:

networkView.txt
packetHandler,h1,10.0.0.1,00:00:00:00:00:01,10.10.1.230,00:02:a0:00:7f:e8,1 host,h2,10.0.0.2,00:00:00:00:00:02,10.10.1.105,00:02:a0:00:73:eb,1 host,h3,10.0.0.3,00:00:00:00:00:03,10.10.2.57,00:02:a0:00:6c:30,1 host,h4,10.0.0.4,00:00:00:00:00:04,10.10.2.85,00:02:a0:00:26:1b,1 host,h5,10.0.0.5,00:00:00:00:00:05,10.10.3.112,00:02:a0:00:9e:25,1 host,h6,10.0.0.6,00:00:00:00:00:06,10.10.3.208,00:02:a0:00:91:68,1

```
host,hp1,10.0.0.5,00:00:00:00:00:05,10.10.1.55,00:02:a0:00:2b:db,1
host,hp2,10.0.0.5,00:00:00:00:00:05,10.10.1.174,00:02:a0:00:7f:db,1
host,hp3,10.0.0.5,00:00:00:00:00:05,10.10.1.196,00:02:a0:00:92:b5,1
host,hp4,10.0.0.5,00:00:00:00:00:05,10.10.2.176,00:02:a0:00:82:f3,1
host,hp5,10.0.0.5,00:00:00:00:00:05,10.10.2.210,00:02:a0:00:83:62,1
host,hp6,10.0.0.5,00:00:00:00:00:05,10.10.2.95,00:02:a0:00:f5:41,1
host,hp7,10.0.0.5,00:00:00:00:00:05,10.10.3.228,00:02:a0:00:78:be,1
host,hp8,10.0.0.5,00:00:00:00:00:05,10.10.3.209,00:02:a0:00:57:a1,1
host,hp9,10.0.0.5,00:00:00:00:00:05,10.10.3.233,00:02:a0:00:bd:4a,1
fakerouter,fr1_if0,10.10.1.1,00:00:00:2b:65:14,1
fakerouter,fr1_if1,10.10.13.1,00:00:00:43:1d:7a,1
fakerouter,fr2_if0,10.10.13.2,00:00:00:93:82:cd,1
fakerouter,fr2_if1,10.10.14.1,00:00:00:83:de:7f,1
fakerouter,fr2_if2,10.10.2.1,00:00:00:c7:5e:02,1
fakerouter,fr3_if0,10.10.3.1,00:00:00:d7:8b:03,1
fakerouter,fr3_if1,10.10.14.2,00:00:00:c2:49:93,1
route,h2,hp1
route,h2,hp2
route,h2,hp3
route,h2,fr1_if0,fr2_if0,h3
route,h2,fr1_if0,fr2_if0,h4
route,h2,fr1_if0,fr2_if0,hp4
route,h2,fr1_if0,fr2_if0,hp5
route,h2,fr1_if0,fr2_if0,hp6
route,h2,fr1_if0,fr2_if0,fr3_if1,h5
route,h2,fr1_if0,fr2_if0,fr3_if1,h6
route,h2,fr1_if0,fr2_if0,fr3_if1,hp7
route,h2,fr1_if0,fr2_if0,fr3_if1,hp8
route,h2,fr1_if0,fr2_if0,fr3_if1,hp9
```

Table 6: Αρχείο κειμένου με την εικονική τοπολογία του δικτύου.

όπου κάθε γραμμή περιέχει και μια οντότητα του δικτύου.

- Αν η οντότητα είναι τύπου packetHandler τότε έχουμε έναν host του mininet ο οποίος τρέχει ένα python script και χειρίζεται τα πακέτα του δικτύου. Η γραμμή που αφορά τον συγκεκριμένο host περιέχει το hostname, την IP διεύθυνση, την MAC διεύθυνση, την virtual IP διεύθυνση, την virtual MAC διεύθυνση και την πόρτα του switch που θα προωθηθούν τα πακέτα που αφορούν τον συγκεκριμένο host.
- Αν η οντότητα είναι τύπου host έχουμε hosts του mininet, π.χ. h2, h3, h4 κλπ και honeypots που αντιστοιχούν σε hosts του mininet, π.χ. hp1, hp2, hp3 κλπ. Η γραμμή που αφορά τον συγκεκριμένο τύπο host περιέχει το hostname, την IP διεύθυνση, την MAC διεύθυνση, την virtual IP διεύθυνση, την virtual MAC διεύθυνση και την πόρτα του switch που θα προωθηθούν τα πακέτα που αφορούν τον συγκεκριμένο host.
- Αν η οντότητα είναι τύπου fakerouter αφορά έναν ψεύτικο router ο οποίος τοποθετείται ανάμεσα στους hosts για να δώσουμε μια ψευδή εικόνα του δικτύου. Η γραμμή που αφορά τον συγκεκριμένο τύπο περιέχει το όνομα ενός interface, μια virtual IP διεύθυνση, μια

virtual MAC διεύθυνση και την πόρτα του switch που θα προωθηθούν τα πακέτα που αφορούν τον συγκεκριμένο host.

- Αν η οντότητα είναι τύπου route τότε αυτή περιέχει τα hops που χρειάζονται για να φτάσουμε από έναν host σε κάποιον άλλο. Για παράδειγμα στο παραπάνω αρχείο παρατηρούμε ότι για να φτάσουμε από τον h2 στον h4 υπάρχουν δύο ενδιάμεσα hops, τα fr1_if0 και fr2_if0 τα οποία ανήκουν σε interfaces των fakerouters.

Εκκίνηση του Ελεγκτή POX

Για να εκκινήσουμε τον ελεγκτή POX αρκεί να εκτελέσουμε την παρακάτω εντολή:

```
>_ sudo /home/mininet/pox/pox.py ibc.iController
```

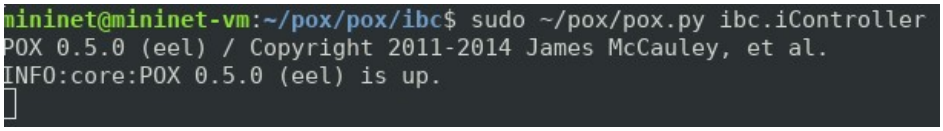


Figure 16: Εκκίνηση του ελεγκτή iController.

Εκκίνηση της τοπολογίας

Για να εκκινήσουμε την τοπολογία στο mininet αρκεί να τρέξουμε το παρακάτω python script:

```
>_ sudo python /home/mininet/myTopology.py
```

Το αποτέλεσμα φαίνεται στην παρακάτω εικόνα

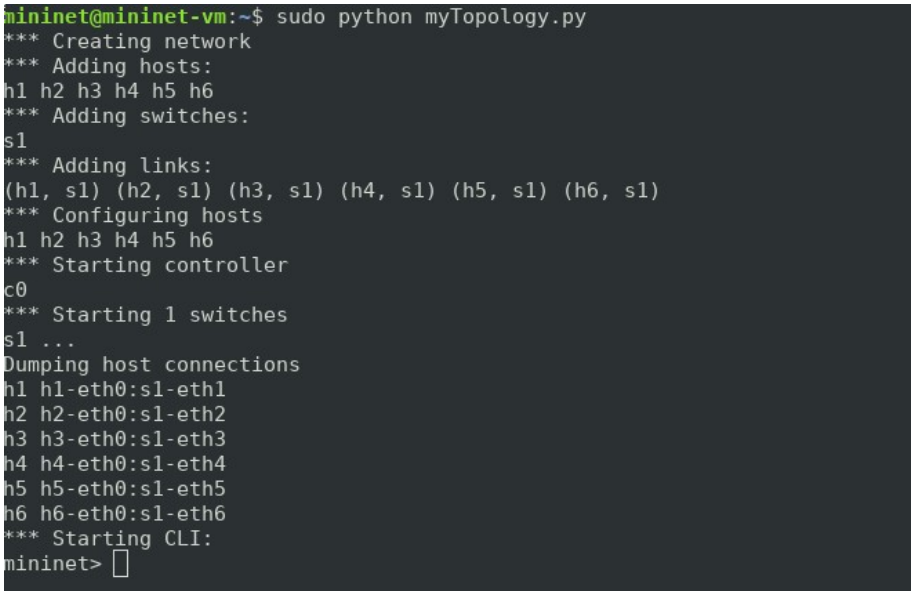


Figure 17: Εκκίνηση της τοπολογίας του mininet

Το παραπάνω python script θα δημιουργήσει μια τοπολογία με την παρακάτω μορφή:

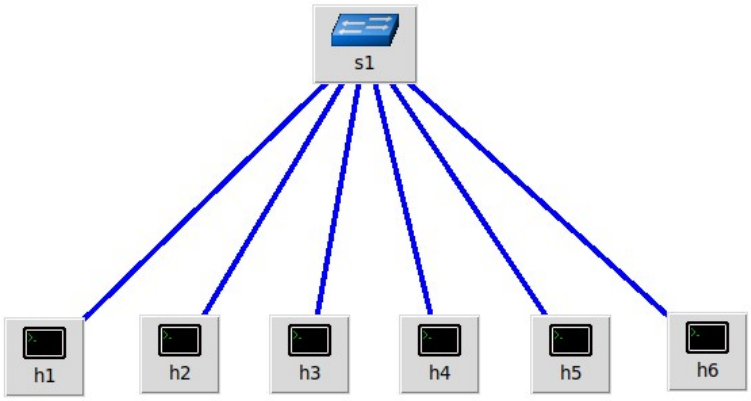


Figure 18: Πραγματική τοπολογία του δικτύου.

Στο σημείο αυτό να αναφέρουμε ότι στο mininet μπορούμε να ανοίξουμε terminal στους hosts χρησιμοποιώντας την εντολή xterm <hostname>. Ένα παράδειγμα ακολουθεί παρακάτω:

Figure 19: Terminal στους virtual hosts του mininet.

Από την πλευρά του ελεγκτή όταν συνδεθούν πάνω του οι hosts της παραπάνω τοπολογίας μας ενημερώνει με σχετικό μήνυμα:

```
mininet@mininet-vm:~/pox/pox/ibc$ sudo ~/pox/pox.py ibc.iController
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Figure 20: Σύνδεση δικτύου με τον controller.

Όταν συνδεθεί μια τοπολογία πάνω στον ελεγκτή τότε αυτός πραγματοποιεί μια σειρά από ενέργειες:

- Δημιουργεί μέσω του createNetworkTopology.py μια νέα εικονική τοπολογία και την αποθηκεύει στο /home/mininet/ibh/networkView.txt ώστε να μπορεί να την προσπελάσει και ο iPacketHandler.
- Διαβάζει την εικονική τοπολογία με τη βοήθεια του networkReader.py
- Ξεκινάει ένα thread το οποίο εκτελείται περιοδικά, π.χ. κάθε 5 λεπτά και αλλάζει τις virtual IP διευθύνσεις και virtual MAC διευθύνσεις της εικονικής τοπολογίας.

```
mininet@mininet-vm:~/pox/pox/ibc$ sudo ~/pox/pox.py ibc.iController
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
['packetHandler,h1,10.0.0.1,00:00:00:00:00:01,10.10.1.28,00:02:a0:00:b9:2b,1',
 'host,h2,10.0.0.2,00:00:00:00:00:02,10.10.1.45,00:02:a0:00:48:c1,1',
 'host,h3,10.0.0.3,00:00:00:00:00:03,10.10.2.72,00:02:a0:00:d3:3b,1',
 'host,h4,10.0.0.4,00:00:00:00:00:04,10.10.2.81,00:02:a0:00:05:01,1',
 'host,h5,10.0.0.5,00:00:00:00:00:05,10.10.3.51,00:02:a0:00:68:4a,1',
 'host,h6,10.0.0.6,00:00:00:00:00:06,10.10.3.90,00:02:a0:00:47:a8,1',
 'host,hp1,10.0.0.5,00:00:00:00:00:05,10.10.1.151,00:02:a0:00:c0:c8,1',
 'host,hp2,10.0.0.5,00:00:00:00:00:05,10.10.1.37,00:02:a0:00:1d:e4,1',
 'host,hp3,10.0.0.5,00:00:00:00:00:05,10.10.1.169,00:02:a0:00:92:ce,1',
 'host,hp4,10.0.0.5,00:00:00:00:00:05,10.10.2.254,00:02:a0:00:95:94,1',
 'host,hp5,10.0.0.5,00:00:00:00:00:05,10.10.2.5,00:02:a0:00:d6:87,1',
 'host,hp6,10.0.0.5,00:00:00:00:00:05,10.10.2.173,00:02:a0:00:05:3c,1',
 'host,hp7,10.0.0.5,00:00:00:00:00:05,10.10.3.155,00:02:a0:00:bd:b7,1',
 'host,hp8,10.0.0.5,00:00:00:00:00:05,10.10.3.207,00:02:a0:00:bc:14,1',
 'host,hp9,10.0.0.5,00:00:00:00:00:05,10.10.3.113,00:02:a0:00:e8:23,1',
 'fakerouter,fr1_if0,10.10.1.1,00:00:00:2b:65:14,1',
 'fakerouter,fr1_if1,10.10.13.1,00:00:00:43:1d:7a,1',
 'fakerouter,fr2_if0,10.10.13.2,00:00:00:93:82:cd,1',
 'fakerouter,fr2_if1,10.10.14.1,00:00:00:83:de:7f,1',
 'fakerouter,fr2_if2,10.10.2.1,00:00:00:c7:5e:02,1',
 'fakerouter,fr3_if0,10.10.3.1,00:00:00:d7:8b:03,1',
 'fakerouter,fr3_if1,10.10.14.2,00:00:00:c2:49:93,1',
 'route,h2,hp1',
 'route,h2,hp2',
 'route,h2,hp3',
 'route,h2,fr1_if0,fr2_if0,h3',
 'route,h2,fr1_if0,fr2_if0,h4',
 'route,h2,fr1_if0,fr2_if0,hp4',
 'route,h2,fr1_if0,fr2_if0,hp5',
 'route,h2,fr1_if0,fr2_if0,hp6',
 'route,h2,fr1_if0,fr2_if0,fr3_if1,h5',
 'route,h2,fr1_if0,fr2_if0,fr3_if1,h6',
 'route,h2,fr1_if0,fr2_if0,fr3_if1,hp7',
 'route,h2,fr1_if0,fr2_if0,fr3_if1,hp8',
 'route,h2,fr1_if0,fr2_if0,fr3_if1,hp9']
```

Figure 21: Ενέργειες ελεγκτή όταν συνδεθεί πάνω του ένα δίκτυο.

Μετά την ανάγνωση του αρχείου /home/mininet/ibh/networkView.txt από τον ελεγκτή και τον iPacketHandler θα μπορούμε να παρουσιάσουμε μια virtual τοπολογία με την παρακάτω μορφή:

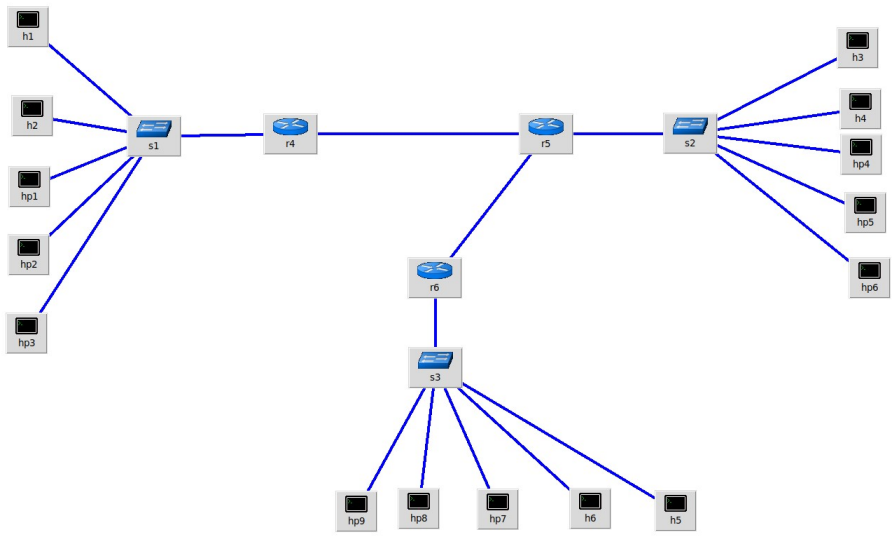


Figure 22: Εικονική τοπολογία του δικτύου.

Λειτουργία του ελεγκτή POX

Στη συνέχεια θα αναλύσουμε πως χειρίζεται τα ARP, ICMP, TCP, UDP ο ελεγκτής POX.

Όπως αναφέραμε και σε προηγούμενο κεφάλαιο, όταν ένα νέο πακέτο φτάσει στο switch αν αυτό δε διαθέτει κάποιον κανόνα για το πώς θα το χειριστεί το στέλνει στον ελεγκτή. Έτσι, όταν ένα πακέτο φτάσει στον ελεγκτή καταγράφεται και επίσης καταγράφεται και η πόρτα του switch από την οποία ήρθε το πακέτο.

Στη συνέχεια στο πακέτο πραγματοποιείται μια σειρά από ελέγχους:

- Αν ο τύπος του πακέτου είναι ARP και το opcode είναι ARP Request ο ελεγκτής στέλνει στο switch δυο κανόνες. Σύμφωνα με τον πρώτο κανόνα το switch πρέπει στείλει το ARP Request στην πόρτα 1 όπου βρίσκεται ο iPacketHandler. Αυτός με τη σειρά του θα δημιουργήσει ARP reply σύμφωνα με το αρχείο networkView.txt και θα το στείλει στο switch. Σύμφωνα με το δεύτερο κανόνα, το switch πρέπει να προωθήσει το ARP Reply στην πόρτα όπου έλαβε το ARP Request. Ο κώδικας για τη δημιουργία των κανόνων είναι ο παρακάτω:

```
if packet.type == packet.ARP_TYPE:
        print(packet)
        if packet.payload.opcode == arp.REQUEST:
            print("ARP Request")

            flowRule = nx.nx_flow_mod()
            flowRule.match.in_port = in_port
            flowRule.match.NXM_OF_ETH_TYPE = ethernet.ARP_TYPE
            flowRule.match.NXM_OF_ARP_OP = arp.REQUEST
            flowRule.match.eth_src = EthAddr(packet.src)
```

```
            flowRule.idle_timeout = 60
            flowRule.priority = 8
            flowRule.actions.append(of.ofp_action_output(port = self.packetHandlerSwitchPort))
            self.connection.send(flowRule)

            currentOutPort = self.packetHandlerSwitchPort
            flowRule = nx.nx_flow_mod()
            flowRule.match.in_port = self.packetHandlerSwitchPort
            flowRule.match.NXM_OF_ETH_TYPE = ethernet.ARP_TYPE
            flowRule.match.NXM_OF_ARP_OP = arp.REPLY
            flowRule.match.eth_dst = EthAddr(packet.src)
            flowRule.idle_timeout = 60
            flowRule.priority = 8
            flowRule.actions.append(of.ofp_action_output(port = in_port))
            self.connection.send(flowRule)

        elif packet.payload.opcode == arp.REPLY:
            print("ARP Reply ")
        else:
            print("Some other ARP opcode.")
```

Table 7: Έλεγχος ARP και κανόνες ροής από τον ελεγκτή.

- Αν ο τύπος του πακέτου είναι IP και το πρωτόκολλο είναι ICMP τότε ο ελεγκτής στέλνει στο switch δυο κανόνες. Σύμφωνα με τον πρώτο κανόνα θα στείλει το ICMP echo request στην πόρτα που βρίσκεται ο iPacketHandler. Αυτός θα δημιουργήσει με τη βοήθεια του scapy ένα echo reply και θα το στείλει στο switch. Σύμφωνα με το δεύτερο κανόνα το switch θα προωθήσει το echo reply στην πόρτα από την οποία έλαβε το echo request. Ο κώδικας των δυο κανόνων παρουσιάζεται παρακάτω:

```
if ip_packet.protocol == ipv4.ICMP_PROTOCOL:
    print("ICMP packet")

    flowRule = nx.nx_flow_mod()
    flowRule.match.of_eth_type = ethernet.IP_TYPE
    flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
    flowRule.match.of_icmp_type = 8 # echo request
    flowRule.match.in_port = in_port
    flowRule.match.eth_src = src_mac
    flowRule.match.ip_src = src_ip
    flowRule.match.ip_dst = dst_ip
    flowRule.match.eth_dst = dst_mac
    flowRule.idle_timeout = 30
    flowRule.priority = 7
    flowRule.actions.append(of.ofp_action_output(port = int(dst_switchPort)))
    self.connection.send(flowRule)

    flowRule = nx.nx_flow_mod()
    flowRule.match.of_eth_type = ethernet.IP_TYPE
    flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
    flowRule.match.of_icmp_type = 0 # echo reply
    flowRule.match.in_port = int(dst_switchPort)
    flowRule.match.eth_src = dst_mac
```

```
flowRule.match.ip_src = dst_ip
flowRule.match.ip_dst = src_ip
flowRule.match.eth_dst = src_mac
flowRule.idle_timeout = 30
flowRule.priority = 7
flowRule.actions.append(of.ofp_action_output(port = in_port))
self.connection.send(flowRule)
```

Table 8: Έλεγχος ICMP και κανόνες ροής του ελεγκτή.

- Αν ο τύπος του πακέτου είναι IP και το πρωτόκολλο είναι TCP τότε ο ελεγκτής στέλνει στο switch δυο κανόνες ανάλογα με το αν ο host από την συμπεριφορά του στο δίκτυο έχει χαρακτηριστεί ως attacker ή όχι.
- Αν ο host έχει χαρακτηριστεί ως attacker τότε ο ελεγκτής στέλνει το switch δυο κανόνες. Σύμφωνα με τον πρώτο κανόνα θα αλλάξει την IP διεύθυνση, τη MAC διεύθυνση και την πόρτα του switch ενός host με τα αντίστοιχα ενός honeypot. Στη συνέχεια αφού απαντήσει το honeypot, ο δεύτερος κανόνας αναλαμβάνει να αλλάξει την IP διεύθυνση, τη MAC διεύθυνση και την πόρτα του switch του honeypot με αυτή του host. Συγκεκριμένα θα εκτελεστούν οι παρακάτω λειτουργίες:
    1. Εάν η διεύθυνση προορισμού είναι ένας host του δικτύου τότε:
       Αλλάζει την IP διεύθυνση προορισμού από την IP διεύθυνση του host στη IP διεύθυνση του honeypot χαμηλής αλληλεπίδρασης.
       Αλλάζει την mac διεύθυνση προορισμού από την mac διεύθυνση του host στη mac διεύθυνση του honeypot χαμηλής αλληλεπίδρασης.
       Προωθεί το πακέτο που προοριζόταν για τον host στην θύρα του switch που είναι συνδεδεμένο το honeypot χαμηλής αλληλεπίδρασης.

    2. Εάν η διεύθυνση προορισμού είναι αυτή του επιτιθέμενου τότε:
       Αλλάζει την IP διεύθυνση του honeypot χαμηλής αλληλεπίδρασης με την IP διεύθυνση του host.
       Αλλάζει την IP διεύθυνση του honeypot χαμηλής αλληλεπίδρασης με την mac διεύθυνση του host.
       Στέλνει το πακέτο στην θύρα του switch που είναι συνδεδεμένος ο επιτιθέμενος.

Ο κώδικας των δυο κανόνων παρουσιάζεται παρακάτω:

```
if src_ip in self.attacker:
   print("Attacker: {}".format(src_ip))

   flowRule = of.ofp_flow_mod()
```

```
flowRule.prioriry = 6
flowRule.match.dl_type = 0x800 # IPv4
flowRule.idle_timeout = 30
#flowRule.hard_timeout = 30
flowRule.match.nw_dst = dst_ip
flowRule.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(self.tcp_honeypot_ip)))
flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(self.tcp_honeypot_mac)))
flowRule.actions.append(of.ofp_action_output(port = self.tcp_honeypot_switchPort))
self.connection.send(flowRule)

flowRule = of.ofp_flow_mod()
flowRule.priority = 6
flowRule.match.dl_type = 0x800
flowRule.idle_timeout = 30
#flowRule.hard_timeout = 30
flowRule.match.nw_dst = IPAddr(src_ip)
flowRule.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(dst_ip)))
flowRule.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(dst_mac)))
flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(src_mac)))
flowRule.actions.append(of.ofp_action_output(port = in_port))
self.connection.send(flowRule)
```

Table 9: Κανόνες ροής TCP για host attacker από τον ελεγκτή.

- Σε περίπτωση που ο host δεν έχει χαρακτηριστεί ως attacker μπορεί να αλληλεπιδράσει με κάποιο host με τους παρακάτω κανόνες:

```
flowRule = of.ofp_flow_mod()
flowRule.prioriry = 6
flowRule.match.dl_type = 0x800 # IPv4
flowRule.idle_timeout = 30
#flowRule.hard_timeout = 30
flowRule.match.nw_dst = dst_ip
flowRule.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(self.net.getHostRealIPAddressFro
mVirtualIPAddress(str(dst_ip)))))
flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(self.net.getHostRealMACAddress
FromVirtualMACAddress(str(dst_mac)))))
flowRule.actions.append(of.ofp_action_output(port = int(real_dst_switchPort)))
self.connection.send(flowRule)

flowRule = of.ofp_flow_mod()
flowRule.priority = 6
flowRule.match.dl_type = 0x800
flowRule.idle_timeout = 30
#flowRule.hard_timeout = 30
flowRule.match.nw_dst = IPAddr(src_ip)
flowRule.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(dst_ip)))
flowRule.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(dst_mac)))
flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(src_mac)))
flowRule.actions.append(of.ofp_action_output(port = in_port))
self.connection.send(flowRule)
```

Table 10: Κανόνες ροής TCP από τον ελεγκτή.

- Αν ο τύπος του πακέτου είναι IP, το πρωτόκολλο είναι UDP και η πόρτα προορισμού είναι η 53 (DNS) τότε ο ελεγκτής στέλνει στο switch δυο κανόνες. Σύμφωνα με τον πρώτο

κανόνα θα στείλει το DNS request στην πόρτα που βρίσκεται ο iPacketHandler. Αυτός θα δημιουργήσει με τη βοήθεια του scapy και του networkView.txt το κατάλληλο DNS reply και θα το στείλει στο switch. Σύμφωνα με το δεύτερο κανόνα το switch θα προωθήσει το DNS reply στην πόρτα από την οποία έλαβε το DNS request. Ο κώδικας των δυο κανόνων παρουσιάζεται παρακάτω:

```
if ip_packet.protocol == ipv4.UDP_PROTOCOL:
   udp_found = packet.find('udp')
      if udp_found.dstport == 53:
         print("UDP packet   DNS")

          flowRule = of.ofp_flow_mod()
          flowRule.prioriry = 5
          flowRule.match.dl_type = 0x800 # IPv4
          flowRule.idle_timeout = 30
          #flowRule.hard_timeout = 0
          flowRule.match.nw_dst = dst_ip

          flowRule.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(self.packetHandlerIP)))
          flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(self.packetHandlerMAC)))
          flowRule.actions.append(of.ofp_action_output(port = int(self.packetHandlerSwitchPort)))
          self.connection.send(flowRule)

          flowRule = of.ofp_flow_mod()
          flowRule.priority = 5
          flowRule.match.dl_type = 0x800
          flowRule.idle_timeout = 30
          #flowRule.hard_timeout = 0
          flowRule.match.nw_dst = IPAddr(src_ip)
          flowRule.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(dst_ip)))
          flowRule.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(dst_mac)))
          flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(src_mac)))
          flowRule.actions.append(of.ofp_action_output(port = in_port))
          self.connection.send(flowRule)
```

Table 11: Έλεγχος UDP στην πόρτα 53 και κανόνες ροής από τον ελεγκτή.

- Αν ο τύπος του πακέτου είναι IP, το πρωτόκολλο είναι UDP και η πόρτα προορισμού είναι μεταξύ των 33434 και 33523 (traceroute) τότε ο ελεγκτής στέλνει στο switch δυο κανόνες. Σύμφωνα με τον πρώτο κανόνα θα στείλει το DNS request στην πόρτα που βρίσκεται ο iPacketHandler. Αυτός θα δημιουργήσει με τη βοήθεια του scapy και του networkView.txt το κατάλληλο DNS reply και θα το στείλει στο switch. Σύμφωνα με το δεύτερο κανόνα το switch θα προωθήσει το ICMP time expired ή ICMP destination/port unreachable στην πόρτα από την οποία έλαβε το DNS request. Ο κώδικας των δυο κανόνων παρουσιάζεται παρακάτω:

```
elif udp_found.dstport >= 33434 and udp_found.dstport <= 33523:
   print("UDP packet   Traceroute")
   flowRule = nx.nx_flow_mod()
   flowRule.match.of_eth_type = ethernet.IP_TYPE
   flowRule.match.of_ip_proto = ipv4.UDP_PROTOCOL
```

```
    flowRule.match.in_port = in_port
    flowRule.idle_timeout = 60
    #flowRule.hard_timeout = 0
    flowRule.priority = 4
    flowRule.actions.append(of.ofp_action_output(port = int(dst_switchPort)))
    self.connection.send(flowRule)

    flowRule = nx.nx_flow_mod()
    flowRule.match.of_eth_type = ethernet.IP_TYPE
    flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
    flowRule.match.of_icmp_type = 11
    flowRule.match.in_port = int(dst_switchPort)
    flowRule.idle_timeout = 60
    #flowRule.hard_timeout = 0
    flowRule.priority = 4
    flowRule.actions.append(of.ofp_action_output(port = in_port))
    self.connection.send(flowRule)

    flowRule = nx.nx_flow_mod()
    flowRule.match.of_eth_type = ethernet.IP_TYPE
    flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
    flowRule.match.of_icmp_type = 3
    flowRule.match.in_port = int(dst_switchPort)
    flowRule.idle_timeout = 0
    flowRule.hard_timeout = 0
    flowRule.priority = 4
    flowRule.actions.append(of.ofp_action_output(port = in_port))
    self.connection.send(flowRule)
```

Table 12: Έλεγχος πόρτας προορισμού 33434 έως 33523 και κανόνες ροής από τον ελεγκτή.

Χαρακτηρισμός ενός Host ως Attacker

Ο ελεγκτής μετά από μια σειρά ελέγχων στα εισερχόμενα πακέτα μπορεί να χαρακτηρίσει κάποιον host ως attacker. Οι έλεγχοι αυτοί αφορούν την αλληλεπίδραση ενός host με κάποιο honeypot και το πλήθος των TCP SYN μηνυμάτων προς πολλές πόρτες ενός host που παραπέμπει σε port scanning, π.χ. χρήση του εργαλείου nmap.

Αλληλεπίδραση host με honeypot με χρήση του εργαλείου ping

Αν η IP διεύθυνση προορισμού ενός echo request είναι η IP διεύθυνση ενός honeypot τότε ο host προστίθεται στην λίστα των attackers που διατηρεί ο ελεγκτής. Ο κώδικας για να το επιτύχουμε είναι ο παρακάτω:

```
if ip_packet.protocol == ipv4.ICMP_PROTOCOL:
    print("ICMP packet")
    if dst_ip in self.honeypots:
        print('ICMP interaction with honeypot {}.'.format(dst_ip))
        if not src_ip in self.attacker:
            self.attacker.append(src_ip)
            print('Add {} as attacker.'.format(src_ip))
```

Table 13: Αλληλεπίδραση host με honeypot μέσω ICMP.

Αλληλεπίδραση host με honeypot με χρήση του εργαλείου traceroute

Αν η IP διεύθυνση προορισμού ενός traceroute είναι η IP διεύθυνση ενός honeypot τότε ο host προστίθεται στην λίστα των attackers που διατηρεί ο ελεγκτής. Ο κώδικας για να το επιτύχουμε είναι ο παρακάτω:

```
elif udp_found.dstport >= 33434 and udp_found.dstport <= 33523:
   print("UDP packet   Traceroute")
   if dst_ip in self.honeypots:
      print('Traceroute interaction with honeypot {}.'.format(dst_ip))
      if not src_ip in self.attacker:
         self.attacker.append(src_ip)
         print('Add {} as attacker.'.format(src_ip))
```

Table 14: Αλληλεπίδραση host με honeypot μέσω traceroute.

Χρήση εργαλείου port scanning

Αν το πλήθος των TCP SYN μηνυμάτων από έναν host προς πολλές πόρτες ενός άλλου host ξεπεράσει το όριο που έχουμε ορίσει τότε ο host προστίθεται στην λίστα των attackers που διατηρεί ο ελεγκτής. Ο κώδικας για να το επιτύχουμε είναι ο παρακάτω:

```
tcp_found = packet.find('tcp')
            if tcp_found:

               appPort = tcp_found.dstport

               if tcp_found.flags == 2: # SYN
                  print "SYN found"

                  if not src_ip in self.knownHosts.keys():
                     newHost = {
                           src_ip:
                           {
                              "SYN_counter": 1,
                              "appPorts": [appPort]
                              }
                           }
                     self.knownHosts.update(newHost)
                  else:
                     for key in self.knownHosts:
                        self.knownHosts[key]['SYN_counter'] = int(self.knownHosts[key]['SYN_counter']) + 1
                        if not appPort in self.knownHosts[key]['appPorts']:
                           self.knownHosts[key]['appPorts'].append(appPort)
                           if ( len(self.knownHosts[key]['appPorts']) > self.appPorts_limit
                                 and
                                 int(self.knownHosts[key]['SYN_counter']) > self.SYN_limit ):

                              if not src_ip in self.attacker:
                                 self.attacker.append(src_ip)
                                 print('Add {} as attacker.'.format(src_ip))

                  print("Known Hosts: {}".format(self.knownHosts))
```

Table 15: Ανίχνευση TCP port scanning.

Δημιουργία νέας τοπολογίας

Όπως αναφέραμε παραπάνω όταν θα συνδεθεί ένα switch στον ελεγκτή POX αμέσως θα δημιουργηθεί μια virtual τοπολογία δικτύου, το αρχείο networkView.txt, με τυχαίες virtual IP και MAC διευθύνσεις. Στη συνέχεια, με τη βοήθεια ενός thread θα ενημερώνονται περιοδικά, π.χ. κάθε 5 λεπτά, οι virtual IP και MAC διευθύνσεις με νέες τυχαίες διευθύνσεις. Σε περίπτωση που υπάρχει μια σύνδεση TCP μεταξύ δυο υπολογιστών αυτή θα διακοπεί όταν αλλάξουν οι virtual IP διευθύνσεις.

```
def __init__ (self, connection, transparent):
        self.connection = connection
        self.transparent = transparent
        connection.addListeners(self)

        self.ct = CreateNetworkTopology()
        self.ct.run()
        filename = self.ct.getFilename()
        print(filename)
        self.net = NetworkReader(filename)
        self.net.readNetworkView()

        self.packetHandlerIP = self.net.getPacketHandlerIP()
        self.packetHandlerMAC = self.net.getPacketHandlerMAC()
        self.packetHandlerSwitchPort = int(self.net.getPacketHandlerSwitchPort())

        self.attacker = []
        self.SYN_counter = 0
        self.SYN_limit = 3

        self.honeypots = self.net.getHoneypots()

        thread.start_new_thread( self.updateTopology, (120, ) )

    def updateTopology(self, delay):
        while True:
            time.sleep(delay)
            self.ct = CreateNetworkTopology()
            self.ct.run()
            filename = self.ct.getFilename()
            print(filename)
            #filename = "/home/mininet/ibh/networkView.txt"
            self.net = NetworkReader(filename)
            self.net.readNetworkView()
```

Table 16: Δημιουργία νέας τοπολογίας.

Χειριστής των πακέτων του δικτύου – iPacketHandler

Ο κώδικας του iPacketHandler βρίσκεται στο directory: /home/mininet/ibh/. Το συγκεκριμένο directory περιέχει τα παρακάτω python scripts:

```
mininet@mininet-vm:~/ibh$ pwd
/home/mininet/ibh
mininet@mininet-vm:~/ibh$ ls -ltr
total 36
-rw-rw-r-- 1 mininet mininet 14274 Ιουν 22 22:08 networkReader.py
-rw-r--r-- 1 root    root     7316 Ιουν 22 22:35 networkReader.pyc
-rw-rw-r-- 1 mininet mininet  7223 Ιουλ  3 19:56 iPacketHandler.py
-rw-r--r-- 1 root    root     1725 Ιουλ 28 12:02 networkView.txt
mininet@mininet-vm:~/ibh$
```

Figure 23: Αρχεία python από τα οποία αποτελείται ο packet handler.

όπου:

- το networkView.txt περιέχει την τοπολογία του δικτύου
- το networkReader.py διαβάζει την τοπολογία του δικτύου από το networkView.txt
- το iPacketHandler.py περιέχει τον κώδικα του χειριστή των πακέτων.

Εκκίνηση του iPacketHandler

Στο terminal του h1 που ανοίξαμε παραπάνω με την εντολή mininet> xterm h1 τρέχουμε το παρακάτω python script:

```
# python ibh/iPacketHandler.py
```

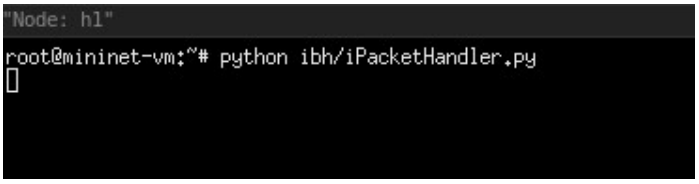


Figure 24: Εκκίνηση του packet handler.

Ο iPacketHandler είναι σε θέση να χειριστεί τα πακέτα συμφώνα με την εικονική τοπολογία που δημιούργησε ο ελεγκτής POX και είναι αποθηκευμένη στο αρχείο networkView.txt. Όπως ο ελεγκτής έτσι και ο iPacketHandler τρέχει περιοδικά ένα thread, π.χ. κάθε 5 λεπτά, το οποίο ενημερώνει την εικονική τοπολογία για τις νέες virtual IP διευθύνσεις και τις virtual MAC διευθύνσεις.

Λειτουργία του iPacketHandler

Ο iPacketHandler με την βοήθεια του scapy διαβάζει τα πακέτα που το switch στέλνει στην πόρτα 1, δηλαδή την πόρτα που βρίσκεται ο host που τρέχει το script iPacketHandler.py. Η κατηγοριοποίηση των πακέτων σε ARP, ICMP, TCP, UDP γίνεται με τον παρακάτω κώδικα.

```
def readPackets(self, pkt):
   #print(pkt.show())
   if pkt[0] not in self.knownPackets:
      if pkt[0].haslayer(ARP):
         print("ARP packet received...")
         self.createARPResponse(pkt)

      if pkt[0].haslayer(ICMP):
         #print(pkt.show())
         print("ICMP packet received...")
         self.createICMPResponse(pkt)

      if pkt[0].haslayer(UDP):
         print("UDP packet received...")
         if pkt[0][UDP].dport == 53:
            print("DNS packet received... ")
            self.handleDNSPacket(pkt)

         if pkt[0].haslayer(IP) and pkt[0][IP].ttl<25:
            print("Send traceroute response...")
            self.createRouteResponse(pkt)
```

Table 17: Έλεγχος πακέτων από τον packet handler.

Στην περίπτωση ενός ARP Request θα δημιουργηθεί ένα ARP Reply με το scapy και θα σταλεί στο switch. Ο iPacketHandler θα βρει την IP διεύθυνση που αντιστοιχεί στη MAC από την ανάγνωση του αρχείου networkView.txt. Ακολουθεί ο κώδικας της συνάρτησης createARPResponse().

```
def createARPResponse(self, pkt):
      if pkt[0].op == 1: # Request
         hw_src = pkt[0].hwsrc
         ip_src = pkt[0].psrc

         ip_dst = pkt[0].pdst
         hw_dst = self.net.getHostVirtualMACAddressFromVirtualIPAddress(ip_dst)
         if hw_dst == None:
            hw_dst = self.getRandomMACAddress()

         arp = eval(pkt[0].command())
         arp[Ether].dst = hw_src
         arp[Ether].src = hw_dst
         arp[ARP].hwdst = hw_src
         arp[ARP].hwsrc = hw_dst
         arp[ARP].pdst = ip_src
         arp[ARP].psrc = ip_dst
         arp[ARP].op = 2

         self.knownPackets.append(arp)
         sendp(arp, inter = 0.001, verbose = 0)
```

Table 18: Δημιουργία και αποστολή απάντησης ARP.

Στην περίπτωση ενός ICMP Echo Request θα δημιουργηθεί ένα ICMP Echo Reply με το scapy και θα σταλεί στο switch. Ακολουθεί ο κώδικας της συνάρτησης createICMPResponse().

```
def createICMPResponse(self, pkt):
    if pkt[0][ICMP].type == 8:
        hw_src = pkt[0][Ether].src
        hw_dst = pkt[0][Ether].dst
        ip_src = pkt[0][IP].src
        ip_dst = pkt[0][IP].dst

        icmp = pkt[0]
        icmp[ICMP].type = 0
        icmp[ICMP].code = 0
        icmp[IP].src = ip_dst
        icmp[IP].dst = ip_src
        icmp[Ether].src = hw_dst
        icmp[Ether].dst = hw_src

        del icmp["ICMP"].chksum

        icmp.show2()
        self.knownPackets.append(icmp)
        sendp(icmp, inter = 0.0001, verbose = 0)
```

Table 19: Δημιουργία και αποστολή απάντησης ICMP.

Στην περίπτωση ενός UDP Request στην πόρτα 53 (DNS query) θα δημιουργηθεί ένα DNS Reply με το scapy και θα σταλεί στο switch. Ο iPacketHandler θα βρει την IP διεύθυνση που αντιστοιχεί στο όνομα του request από την ανάγνωση του αρχείου networkView.txt. Ακολουθεί ο κώδικας της συνάρτησης handleDNSPacket().

```
def handleDNSPacket(self, pkt):
    #print(pkt.show())
    dp=pkt[0][IP][UDP].dport
    sp=pkt[0][IP][UDP].sport

    ip_src = pkt[0][IP].src
    ip_dst = pkt[0][IP].dst

    hw_src = pkt[0][Ether].src
    hw_dst = pkt[0][Ether].dst

    q = pkt[0][IP][DNS][DNSQR].qname
    qn = q.split('.')
    if len(qn) == 2:    # DNS lookup
        print("dns lookup {}".format(qn[0]))
        ipaddr = self.net.getHostVirtualIPAddressFromName(str(qn[0]))
    elif len(qn) >= 6:  # reverse DNS lookup
        ip = qn[3] + '.' + qn[2] + '.' + qn[1] + '.' +qn[0]
        print("reverse DNS lookup {}".format(ip))
        ipaddr = ip
        return
```

```
    # ipaddr = self.net.getHostVirtualIPAddressFromName('h3')
    #print("ip_src {}".format(ip_src))
    #print("ip_dst {}".format(ip_dst))
    #print("hw_src {}".format(hw_src))
    #print("hw_dst {}".format(hw_dst))
    #print("ipaddr {}".format(ipaddr))

    ether = Ether(src = hw_dst, dst = hw_src)
    ip = IP(dst = ip_src, src = ip_dst)
    udp = UDP(sport = dp, dport = sp)
    dns = DNS(
          qr = 1,
          id = pkt[0][IP][DNS].id,
          qd = DNSQR(qname=pkt[0][IP][DNS][DNSQR].qname),
          an = (
             DNSRR(rrname=pkt[0][IP][DNS][DNSQR].qname,
             rdata = ipaddr,
             ttl = 3600,
             rclass = "IN",
             type = "A")
             ),
             ar = 1) #DNSRROPT(rclass=3000)

    dns_resp = ether/ip/udp/dns
    dns_resp.show2()
    self.knownPackets.append(dns_resp)
    sendp(dns_resp, inter=0.001, verbose=0)
```

Table 20: Δημιουργία και αποστολή απάντησης DNS.

Στην περίπτωση ενός UDP πακέτου με TTL μικρότερο από 25 (traceroute) το scapy και θα σταλεί στο switch την αντίστοιχη ICMP (time expired ή destination/ port unreachable) απάντηση ανάλογα με τα hops που υπάρχουν από τον source host στον destination host . Ο iPacketHandler θα βρει ενδιάμεσα hops από την ανάγνωση του αρχείου networkView.txt. Ακολουθεί ο κώδικας της συνάρτησης createRouteResponse().

```
def createRouteResponse(self, pkt):
    src_eth = pkt[0][Ether].src
    dst_eth = pkt[0][Ether].dst
    src_ip = pkt[0][IP].src
    dst_ip = pkt[0][IP].dst
    cur_ttl = pkt[0][IP].ttl

    route = self.net.getRoute(str(src_ip), str(dst_ip))
    if cur_ttl <= len(route) - 1:

        hop_ip = route[cur_ttl]
        if hop_ip != dst_ip:
            hop_eth = self.net.getFakerouterVirtualMACAddressFromIP(hop_ip)
            #icmp time exceed during transmission
            ether = Ether( src = hop_eth, dst = src_eth)
            ip = IP(src = hop_ip, dst = src_ip)
            icmp = ICMP(type = 11, code = 0)
            resp = ether/ip/icmp/IPerror(str(pkt[0][IP]))
```

```
            #recalculate checksum
            resp.show2()
            if hop_ip == dst_ip:
            #icmp destination and port unreachable
            hop_eth = self.net.getHostVirtualMACAddressFromVirtualIPAddress(hop_ip)
            ether = Ether(src = hop_eth, dst = src_eth)
            ip = IP(src = hop_ip, dst = src_ip)
            icmp = ICMP(type=3, code=3)
            resp = ether/ip/icmp/IPerror(str(pkt[0][IP]))
            #recalculate checksum
            resp.show2()

        self.knownPackets.append(resp)
        sendp(resp, inter=0.001, verbose=0)
```

Table 21: Δημιουργία και αποστολή απάντησης traceroute.

Όπως έχουμε ήδη αναφέρει ο iPackerHandler διαβάζει περιοδικά, π.χ. κάθε 5 λεπτά, τις νέες virtual IP και MAC διευθύνσεις με τη βοήθεια ενός thread. Ο κώδικας για να το επιτύχουμε είναι ο παρακάτω:

```
def __init__(self, filename):
    self.knownPackets = []
    self.filename = filename
    self.net = NetworkReader(self.filename)
    self.net.readNetworkView()

    thread.start_new_thread( self.updateTopology, (300, ) )

  def updateTopology(self, delay):
    while True:
      time.sleep(delay)
      filename = "/home/mininet/ibh/networkView.txt"
      self.net = NetworkReader(filename)
      self.net.readNetworkView()
      print("Update topology.")
```

Table 22: Ανανέωση τοπολογίας από τον packet handler.

Σενάριο εφαρμογής του μηχανισμού παραπλάνησης

Στη συνέχεια θα παρουσιάσουμε ένα σενάριο εφαρμογής του μηχανισμού παραπλάνησης που περιγράψαμε στο προηγούμενο κεφάλαιο. Αρχικά, υποθέτουμε ότι στον υπολογιστή h2 έχει πρόσβαση ο επιτιθέμενος και θα προσπαθήσει να συλλέξει πληροφορίες για τους υπόλοιπους υπολογιστές του δικτύου καθώς και για την τοπολογία του δικτύου.

Αν ο επιτιθέμενος από τον h2 εκτελέσει την εντολή `ping h3` θα πάρει την παρακάτω απάντηση:

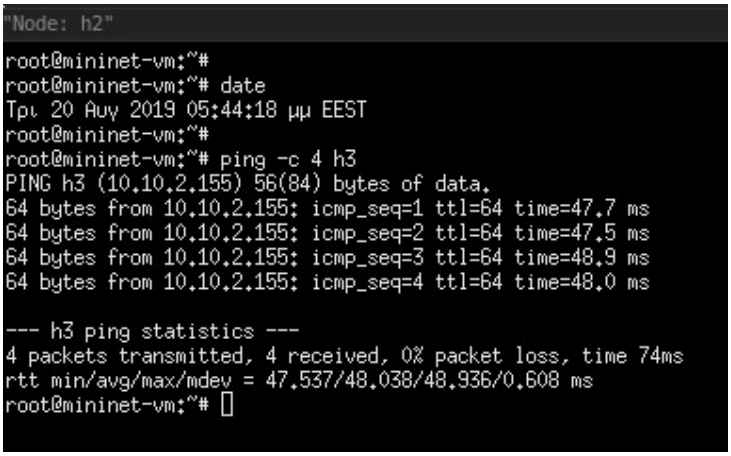


Figure 25: Ping σε host με χρήση του hostname.

Παρατηρούμε ότι από την απάντηση που έλαβε από τον h3 έχει διεύθυνση IP 10.10.2.155, δηλαδή παρουσιάζεται η virtual IP διεύθυνση 10.10.2.155 και όχι η πραγματική 10.0.0.3. Όπως αναφέραμε και στο προηγούμενο κεφάλαιο η virtual IP διεύθυνση καθώς και η virtual MAC διεύθυνση ανανεώνονται κάθε πέντε λεπτά. Αν στην συνέχεια ο επιτιθέμενος προσπαθήσει και πάλι να εκτελέσει την εντολή `ping h3` θα λάβει τα παρακάτω αποτελέσματα:

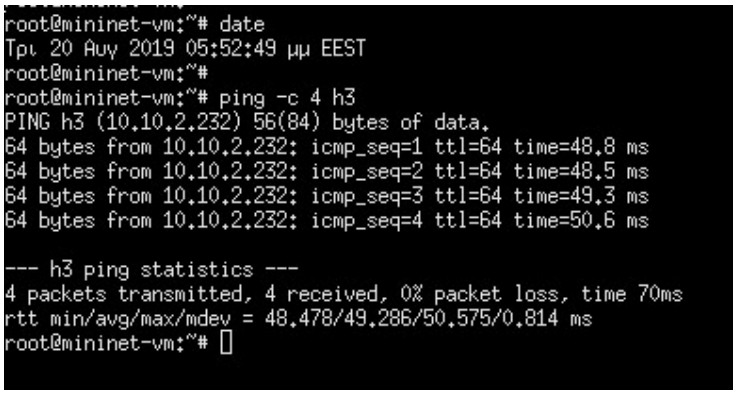


Figure 26: Ping σε host με χρήση του hostname μετά την ανανέωση της τοπολογίας.

Όπως παρατηρούμε μετά από πέντε λεπτά η IP διεύθυνση που παρουσιάζεται στον επιτιθέμενο είναι η 10.10.2.232.

Αν ο επιτιθέμενος προσπαθήσει να αλληλεπιδράσει με κάποιο από τα honeypots του δικτύου με ICMP (π.χ. με την εντολή ping) αμέσως ο Controller θα τον χαρακτηρίσει ως attacker. Ακολουθεί παράδειγμα της συγκεκριμένης περίπτωσης:

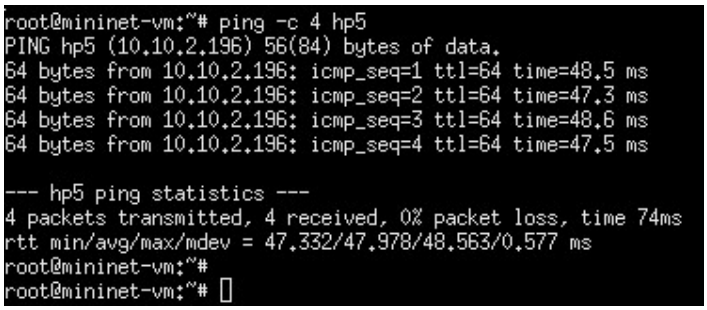


Figure 27: Ping σε honeypot με χρήση του hostname.

Το hp5 (honeypot 5) απαντάει νε την virtual IP διεύθυνση 10.10.2.196 και αυτομάτως ο Controller τον χαρακτηρίζει ως attacker, όπως παρατηρούμε στην παρακάτω εικόνα:

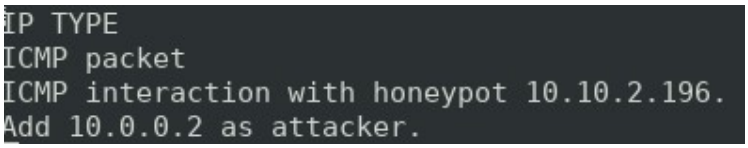


Figure 28: Χαρακτηρισμός του host ως attacker μετά την αλληλεπίδραση με το honeypot.

Μια ακόμα ενέργεια που μπορεί να κάνει ο επιτιθέμενος είναι να χρησιμοποιήσει το πρόγραμμα nmap ώστε να εντοπίσει της πόρτες (εφαρμογές) που έχει ανοικτές κάποιος host. Αυτή την ενέργεια μπορεί να την εντοπίσει ο Controller και να χαρακτηρίσει τον host από τον οποίο ξεκίνησε το nmap ως attacker. Τα κριτήρια για να εντοπίσει μια τέτοια επίθεση ο Controller είναι το πλήθος των SYN μηνυμάτων που στέλνει ο συγκεκριμένος host, καθώς και το πλήθος των διαφορετικών πορτών που έχουν στόχο τα SYN μηνύματα. Αν δεν λαμβάναμε υπόψη το πλήθος των διαφορετικών πορτών υπήρχε κίνδυνος να χαρακτηριστεί ως attacker ένας host του δικτύου που θα ζητούμε μια υπηρεσία του δικτύου πολλές φορές (π.χ. μια web υπηρεσία).
Στη συνέχεια ακολουθεί ένα παράδειγμα όπου ο επιτιθέμενος από τον h2 κάνει scan τις υπηρεσίες του h3.

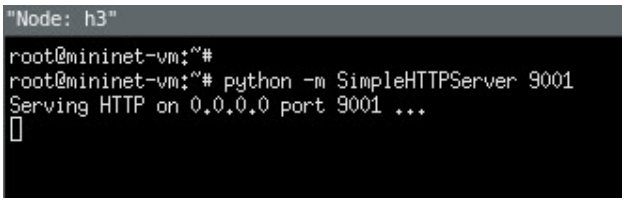


Figure 29: O host h3 λειτουργεί ως HTTP Server.

Ο host h3 τρέχει ένα http service στην πόρτα 9001. Ο επιτιθέμενος από τον h2 τρέχει την εντολή `nmap 10.10.2.200`. Την IP διεύθυνση αναφέραμε σε προηγούμενα βήματα πως μπορούμε να την βρούμε.

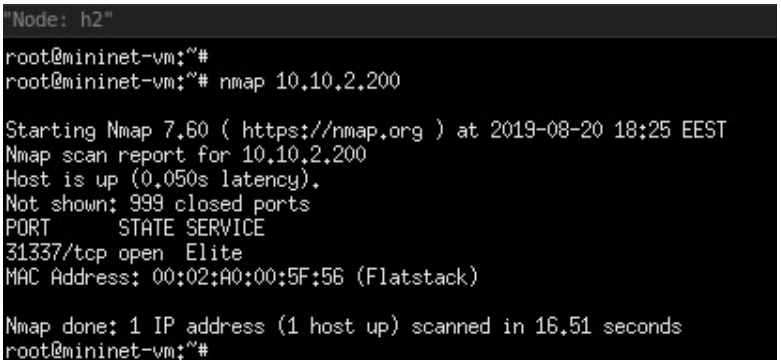


Figure 30: O host h2 πραγματοποιεί TCP port scanning στον h3.

Όπως παρατηρούμε στην παραπάνω εικόνα, ο h3 παρουσιάζεται να έχει ανοικτή την πόρτα 31337 ενώ παραπάνω είδαμε ότι έχει ανοικτή την πόρτα 9001. Αυτό συνέβη επειδή ο Controller εντόπισε ένα πλήθος μηνυμάτων SYN προς πολλές πόρτες του host h3. Στη συνέχεια ο Controller έστειλε την κίνηση που προοριζόταν για τον h3 στον h5 ο οποίος έχει τον ρόλο του TCP honeypot στο δίκτυο. Η παρακάτω εικόνα παρουσιάζει τα logs του Controller και πως αυτός εντόπισε την επίθεση.

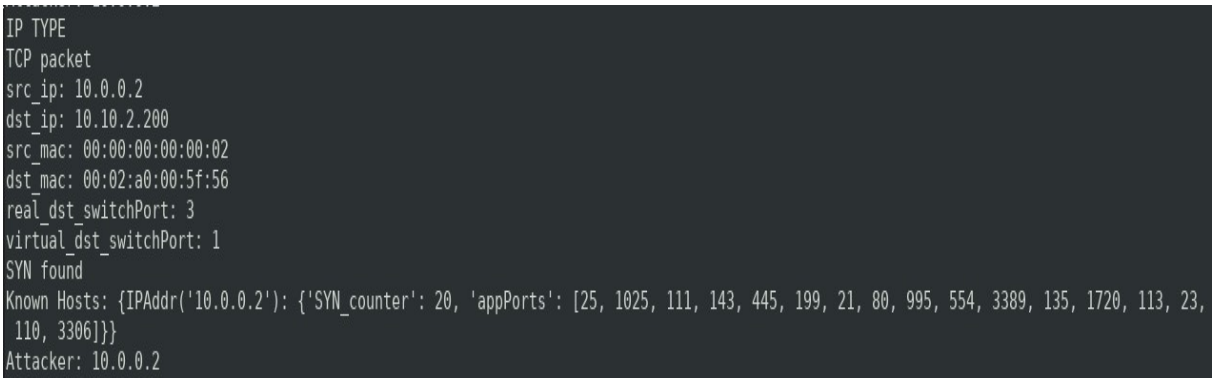


Figure 31: O h2 χαρακτηρίζεται ως attacker από τον ελεγκτή λόγω του port scanning.

Η παρακάτω εικόνα παρουσιάζει τον host h5, ο οποίος αναφέραμε ότι έχει τον ρόλο του TCP honeypot στο δίκτυο. Ο συγκεκριμένος host έχει ανοίξει την πόρτα 31337 με την βοήθεια της εντολής netcat, δηλαδή `nc -l -p 31337`.

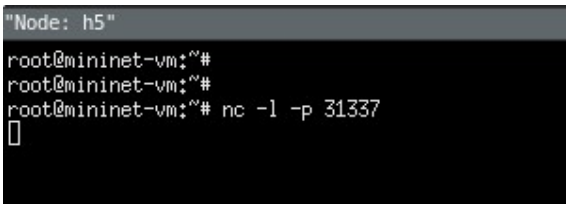


Figure 32: Ο h5 λειτουργεί ως honeypot.

Ο host h5 μπορεί να λειτουργεί σαν ένα honeypot χαμηλής αλληλεπίδρασης όταν εκτελείται το honeypot.py

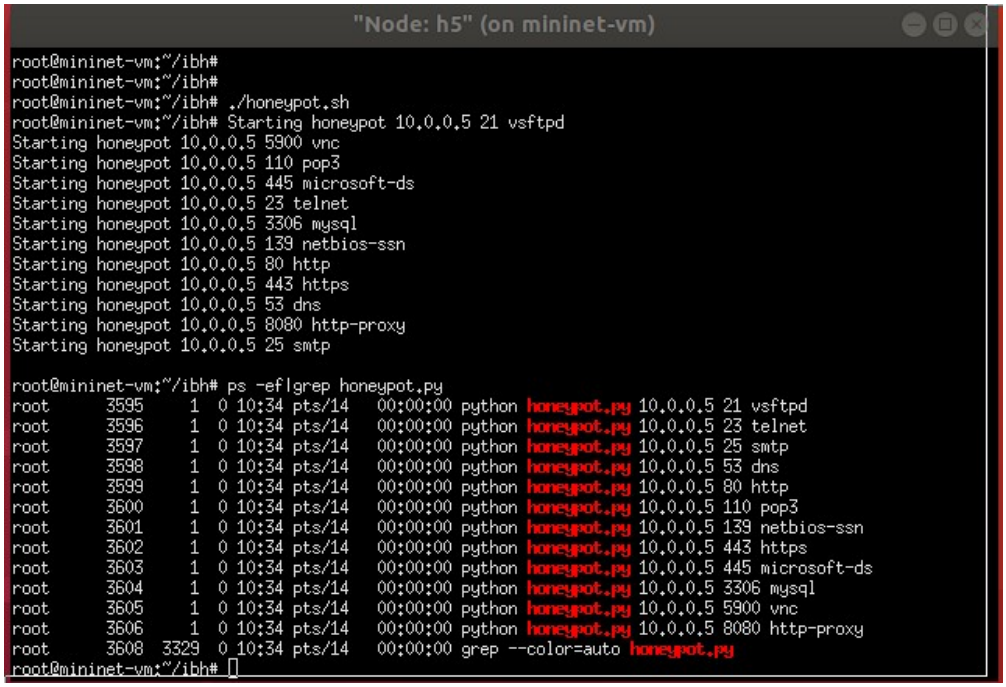


Figure 33: Host h5 ως honeypot χαμηλής αλληλεπίδρασης.

Ο επιτιθέμενος μετά από σάρωση στο δίκτυο με το nmap θα έχει την παρακάτω εικόνα.

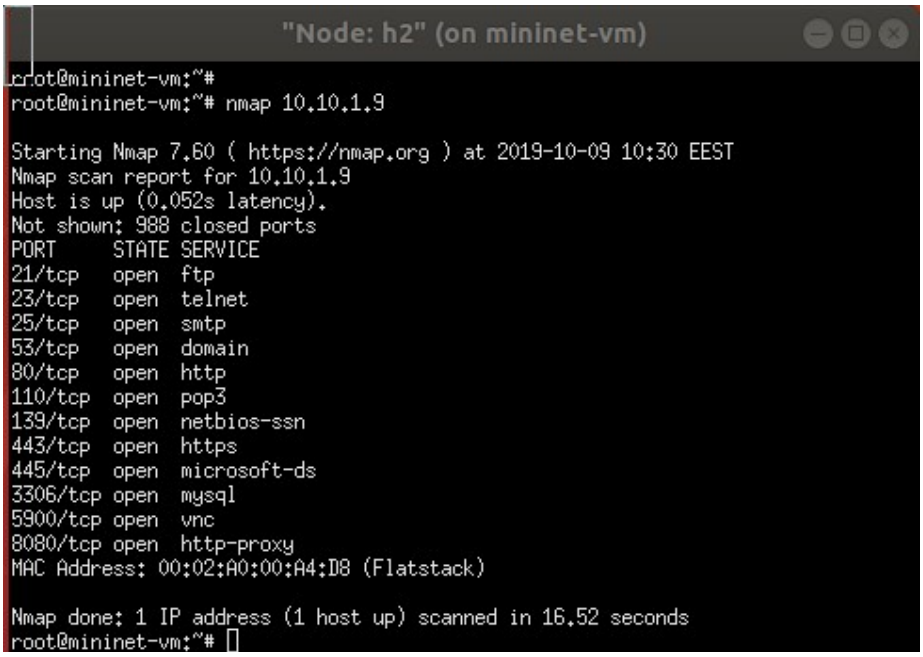


Figure 34: nmap στο honeypot χαμηλής αλληλεπίδρασης.

Στη συνέχεια θα δούμε πως συμπεριφέρεται ο Controller όταν κάποιος host ζητήσει μια νόμιμη υπηρεσία, για παράδειγμα να αποθηκεύσει μια ιστοσελίδα τοπικά με τη βοήθεια του προγράμματος wget.

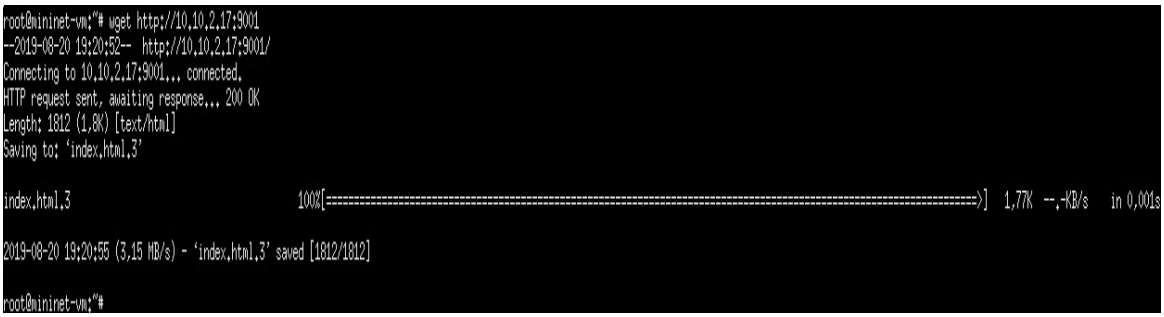


Figure 35: Κανονική αλληλεπίδραση με τον web server.

Όπως παρατηρούμε ο συγκεκριμένος host μπόρεσε να αποθηκεύσει την ιστοσελίδα τοπικά χωρίς προβλήματα.

Μια ακόμη ενέργεια που μπορεί να κάνει ο επιτιθέμενος είναι να εντοπίσει το πλήθος των ενδιάμεσων δικτύων (hops) ανάμεσα σε δυο υπολογιστές, με σκοπό να χαρτογραφήσει το δίκτυο. Το παραπάνω μπορεί να το πραγματοποιήσει με την βοήθεια της εντολής `traceroute <ip address>`.

Στη συνέχεια, ο επιτιθέμενος θα εκτελέσει μια σειρά από εντολές `traceroute` από τον host h2.

- Traceroute προς τον 10.10.2.251 (host h3):

```
root@mininet-vm:~# traceroute 10.10.2.251
traceroute to 10.10.2.251 (10.10.2.251), 30 hops max, 60 byte packets
 1  10.10.1.1 (10.10.1.1)  57.806 ms  59.783 ms  67.836 ms
 2  10.10.13.2 (10.10.13.2)  73.620 ms  80.877 ms  94.759 ms
 3  10.10.2.251 (10.10.2.251)  97.037 ms  105.125 ms  109.043 ms
root@mininet-vm:~#
```

- Traceroute προς τον 10.10.2.138 (host h4):

```
root@mininet-vm:~# traceroute 10.10.2.138
traceroute to 10.10.2.138 (10.10.2.138), 30 hops max, 60 byte packets
 1  10.10.1.1 (10.10.1.1)  96.021 ms  101.862 ms  108.742 ms
 2  10.10.13.2 (10.10.13.2)  115.614 ms  121.560 ms  134.290 ms
 3  10.10.2.138 (10.10.2.138)  140.752 ms  146.740 ms  153.083 ms
root@mininet-vm:~#
```

- Traceroute προς τον 10.10.2.33 (host hp4):

```
root@mininet-vm:~# traceroute 10.10.2.33
traceroute to 10.10.2.33 (10.10.2.33), 30 hops max, 60 byte packets
 1  10.10.1.1 (10.10.1.1)  98.732 ms  104.051 ms  109.527 ms
 2  10.10.13.2 (10.10.13.2)  117.047 ms  122.389 ms  127.507 ms
 3  10.10.2.33 (10.10.2.33)  137.370 ms  142.945 ms  148.640 ms
root@mininet-vm:~#
```

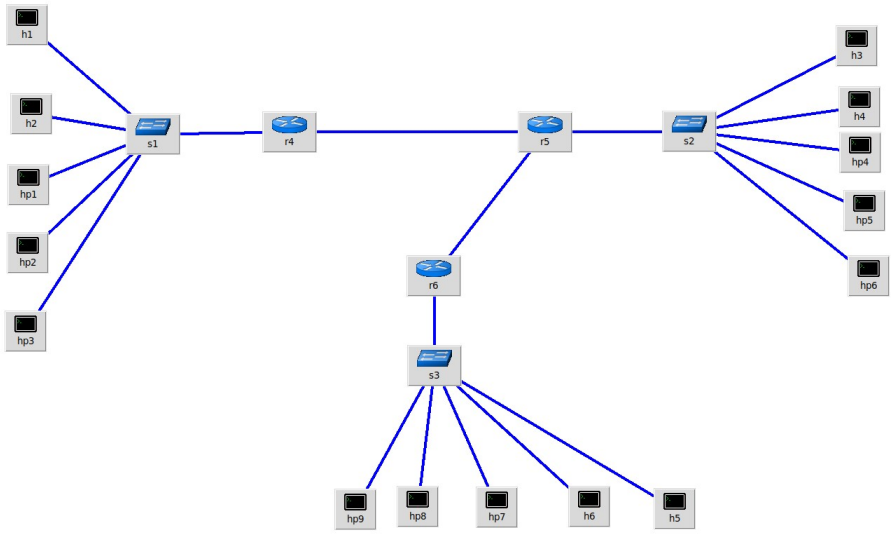
- Traceroute προς τον 10.10.2.19 (host hp5):

```
root@mininet-vm:~# traceroute 10.10.2.19
traceroute to 10.10.2.19 (10.10.2.19), 30 hops max, 60 byte packets
 1  10.10.1.1 (10.10.1.1)  102.499 ms  108.772 ms  115.397 ms
 2  10.10.13.2 (10.10.13.2)  122.296 ms  130.051 ms  135.696 ms
 3  10.10.2.19 (10.10.2.19)  146.862 ms  215.394 ms  221.855 ms
root@mininet-vm:~#
```

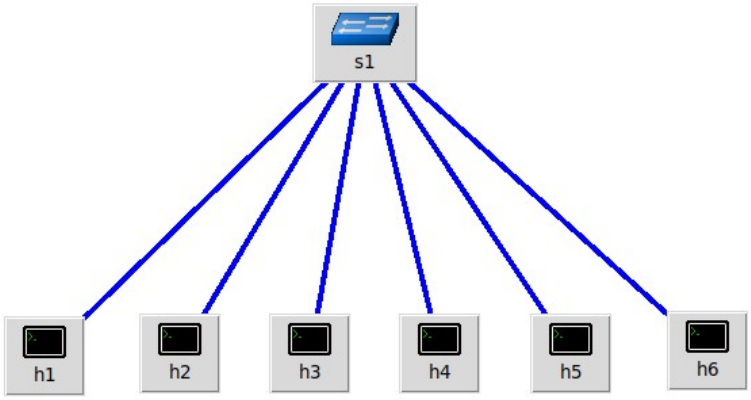
- Traceroute προς τον 10.10.2.144 (host hp6):

```
root@mininet-vm:~# traceroute 10.10.2.144
traceroute to 10.10.2.144 (10.10.2.144), 30 hops max, 60 byte packets
 1  10.10.1.1 (10.10.1.1)  97.525 ms  102.473 ms  108.273 ms
 2  10.10.13.2 (10.10.13.2)  115.384 ms  123.972 ms  136.249 ms
 3  10.10.2.144 (10.10.2.144)  142.862 ms  149.975 ms  157.088 ms
root@mininet-vm:~#
```

Αν ο επιτιθέμενος συνεχίσει να εργάζεται με τον παραπάνω τρόπο προς όλους τους host του δικτύου θα έχει την παρακάτω εικόνα για την τοπολογία του δικτύου:



Ενώ η πραγματική εικόνα του δικτύου είναι η παρακάτω:



Όπως είδαμε παραπάνω ο επιτιθέμενος μέσω της εντολής traceroute αλληλεπίδρασε με τα honeypots hp4, hp5 και hp6. Ο Controller παρακολουθώντας τη συγκεκριμένη δραστηριότητα χαρακτηρίζει τον host h2 ως attacker.

```
IP TYPE
UDP packet   Traceroute
Traceroute interaction with honeypot 10.10.2.156.
Add 10.0.0.2 as attacker.
```

Figure 36: Αλληλεπίδραση με honeypot μέσω traceroute και χαρακτηρισμός ως attacker.

Στη συνέχεια θα παρατηρήσουμε πως συμπεριφέρεται το δίκτυο στην περίπτωση που επιτιθέμενος πραγματοποιήσει ping sweep στο δίκτυο με την βοήθεια του προγράμματος nmap.

```
root@mininet-vm:~# nmap -sP 10.10.2.0-254

Starting Nmap 7.60 ( https://nmap.org ) at 2019-08-20 19:26 EEST
Nmap scan report for 10.10.2.47
Host is up (-0.13s latency).
MAC Address: 00:02:A0:00:21:1C (Flatstack)
Nmap scan report for 10.10.2.139
Host is up (0.063s latency).
MAC Address: 00:02:A0:00:26:EF (Flatstack)
Nmap scan report for 10.10.2.140
Host is up (0.065s latency).
MAC Address: 00:02:A0:00:7F:86 (Flatstack)
Nmap scan report for 10.10.2.227
Host is up (-0.14s latency).
MAC Address: 00:02:A0:00:7F:09 (Flatstack)
Nmap scan report for 10.10.2.244
Host is up (-0.14s latency).
MAC Address: 00:02:A0:00:A0:34 (Flatstack)
Nmap done: 255 IP addresses (5 hosts up) scanned in 19.12 seconds
root@mininet-vm:~#
```

Figure 37: Nmap ping sweep.

Παρατηρούμε ότι και πάλι ο επιτιθέμενος έχει την εικονική εικόνα του δικτύου που του παρουσιάζεται από τις virtual IP διευθύνσεις των hosts.

Συμπεράσματα

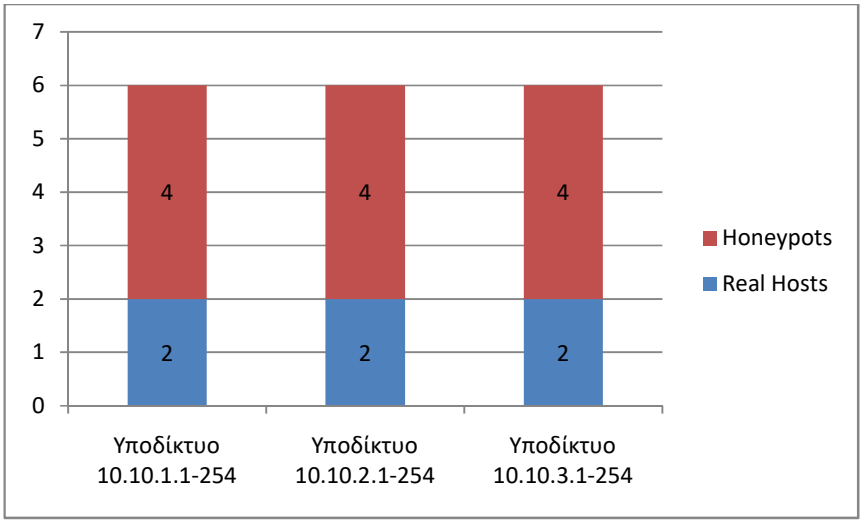
Τα παραδοσιακά δίκτυα είναι πολύπλοκα και δύσκολα διαχειρίζονται την ασφάλεια στον κυβερνοχώρο. Η τεχνολογία SDN και οι τεχνικές αμυντικής παραπλάνησης θα μπορούσαν να διευκολύνουν στον εντοπισμό δυνητικών επιθέσεων στον κυβερνοχώρο και στην προστασία των δικτύων υπολογιστών. Επιπλέον, οι περισσότερες συσκευές δικτύωσης συνοδεύονται με εγκατεστημένο διαφορετικό υλικό και λογισμικό ανάλογα με τον κατασκευαστή. Για το σκοπό αυτό, ο χειρισμός τέτοιων συσκευών αποτελεί πρόκληση και απαιτεί μεγάλη προσπάθεια η εξασφάλιση των λειτουργιών τους. Χρησιμοποιώντας την τεχνολογία SDN και διάφορες τεχνικές αμυντικής παραπλάνησης, όπως περιγράψαμε και στο αντίστοιχο κεφάλαιο, μπορούν να κατασκευαστούν ασφαλή και κλιμακούμενα δίκτυα.

Όπως φαίνεται από τον πίνακα 2, μόνο τέσσερις από τις δέκα τεχνικές αμυντικής παραπλάνησης έχουν υλοποιηθεί χρησιμοποιώντας την τεχνολογία SDN. Ειδικότερα, έχει καταβληθεί μεγάλη προσπάθεια για την ανάπτυξη τεχνικών MTD με την αξιοποίηση της τεχνολογίας SDN, δεδομένου ότι αυτή η τεχνική καθίσταται πιο αποτελεσματική όσο επεκτείνεται το περιβάλλον του δικτύου. Επιπλέον, η δυνατότητα προγραμματισμού των συσκευών δικτύου διευκολύνει τη ροή δεδομένων μέσω του δικτύου, εφόσον μπορεί να επιτευχθεί η εφαρμογή διαφορετικών πολιτικών. Επιπλέον, τρείς από τις δεκατέσσερις από τις εργασίες που εξετάσαμε πρότειναν την εφαρμογή των Honeypots ή Honeytokens στα SDN. Παρόλο που αυτή η αμυντική τεχνική είναι γνωστή στις αρχιτεκτονικές των παραδοσιακών δικτύων, η υλοποίησή της δεν είναι αποδοτική, αφού οι εισβολείς μπορούν να την εντοπίσουν εύκολα. Ωστόσο, χρησιμοποιώντας την τεχνολογία SDN, τα honeypots θα μπορούσαν να μην είναι ανιχνεύσιμα και επομένως να προστατεύουν την υποδομή δικτύου από επιθέσεις στον κυβερνοχώρο, όπως η σάρωση δικτύου ή μια επίθεση DDoS. Από την άλλη πλευρά, μόνο δύο εργασίες έχουν αναλύσει την εφαρμογή των υπηρεσιών Decoy και των παραπλανητικών τεχνολογιών χρησιμοποιώντας ελεγκτές SDN.

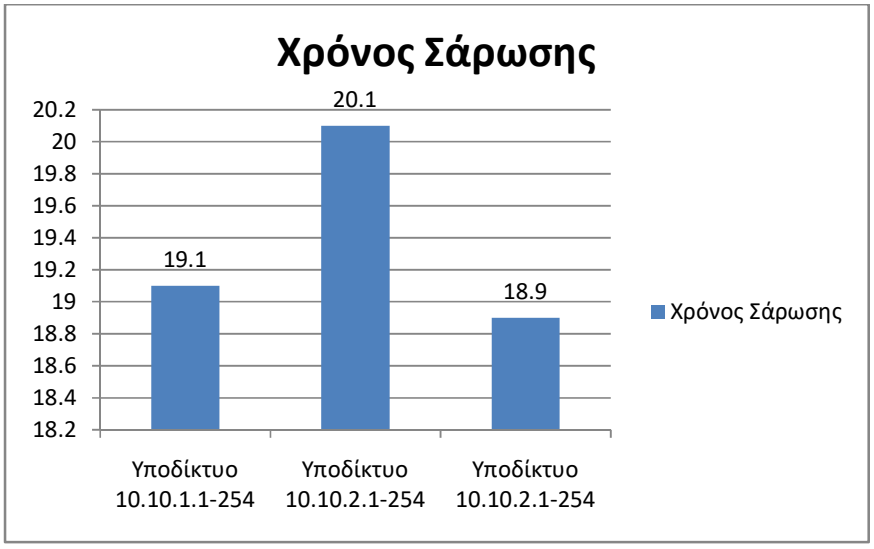
Ενώ οι αμυντικές τεχνικές παραπλάνησης είναι σε θέση να προστατεύουν μεγάλα και σύνθετα δίκτυα χρησιμοποιώντας την τεχνολογία SDN, πολλές από αυτές τις τεχνικές δεν έχουν ακόμη εξεταστεί. Περαιτέρω, τεχνικές όπως το Netwrok Tarpit και το OS obfuscation θα μπορούσαν να αναπτυχθούν χρησιμοποιώντας την τεχνολογία SDN. Αξιοποιώντας τις δυνατότητες δυναμικού προγραμματισμού του SDN, είναι εφικτό ένα μιμητικό δίκτυο ή μια καθυστέρηση στην κίνηση μέσω του δικτύου. Αν και ορισμένες από τις αμυντικές τεχνικές, όπως το traffic forging και η παραπλανητική προσομοίωση, θα μπορούσαν να έχουν παρόμοια αποτελέσματα με εκείνες που έχουν ήδη αναλυθεί, θα ήταν χρήσιμο να αναλυθούν ξεχωριστά και να προσδιοριστούν τα πλεονεκτήματα και τα μειονεκτήματα προκειμένου να διεξαχθεί μια πιο ολοκληρωμένη έρευνα για την ολοκλήρωση το SDN σε τέτοιες τεχνικές.

Στη συνέχεια, θα παρουσιάσουμε τα αποτελέσματα σαρώσεων που πραγματοποιήθηκαν με την χρήση του εργαλείου nmap (εντολή `nmap -sP 10.10.1.1-254`) σε ένα δίκτυο που χρησιμοποιεί τον μηχανισμό αμυντικής παραπλάνησης που παρουσιάσαμε στο πρακτικό μέρος της εργασίας.

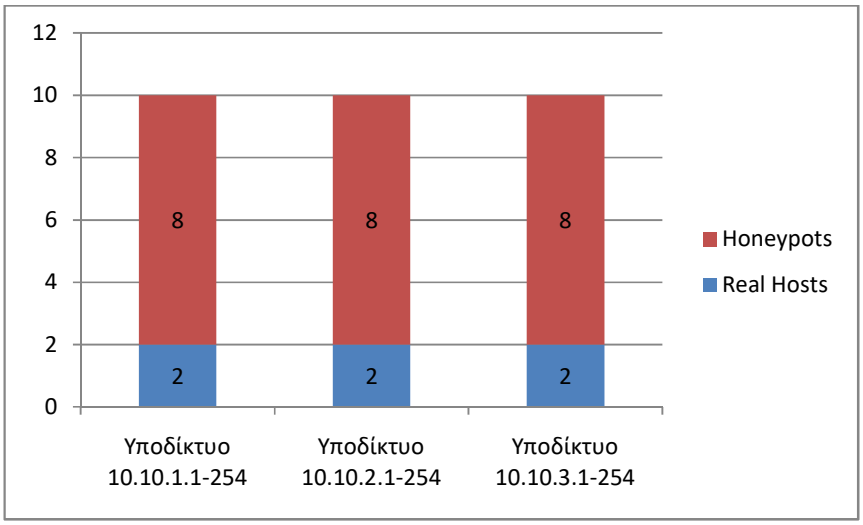
Για ένα δίκτυο το οποίο περιέχει δεκαοκτώ υπολογιστές από τους οποίους οι έξι είναι πραγματικοί hosts, οι δώδεκα είναι honeypots και υπάρχουν τρία υποδίκτυα υπολογιστών έχουμε τα παρακάτω γραφήματα.



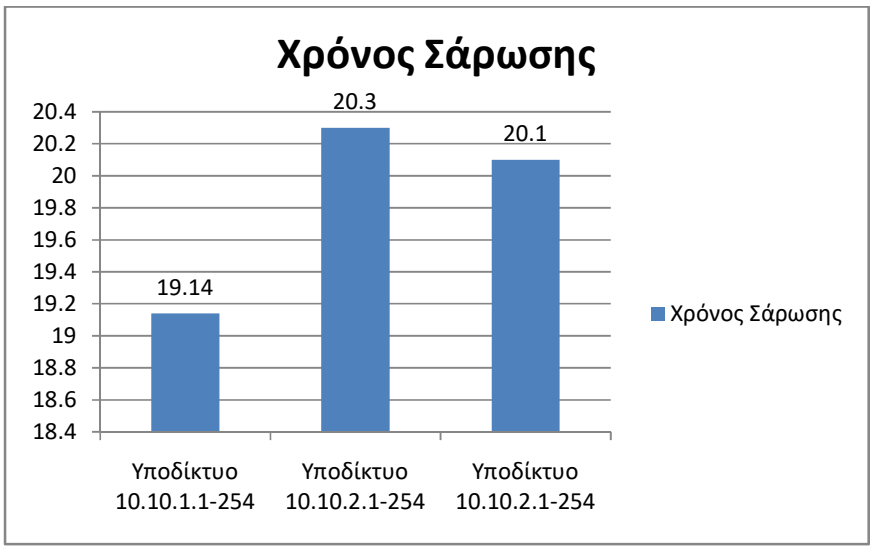
Ο συνολικός χρόνος που απαιτείται για την σάρωση όλων των υποδίκτυων είναι 57.3 δευτερόλεπτα.



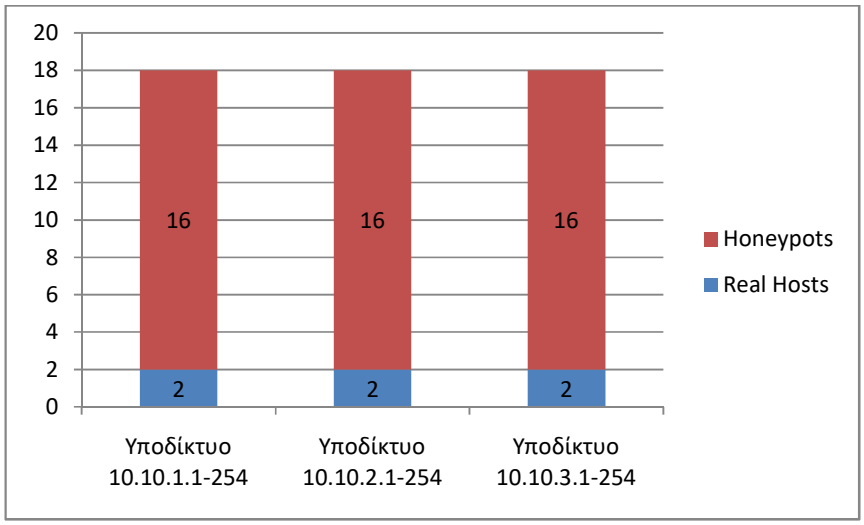
Για ένα δίκτυο το οποίο περιέχει τριάντα υπολογιστές από τους οποίους οι έξι είναι πραγματικοί hosts, οι εικοσιτέσσερις είναι honeypots και υπάρχουν τρία υποδίκτυα υπολογιστών έχουμε τα παρακάτω γραφήματα.



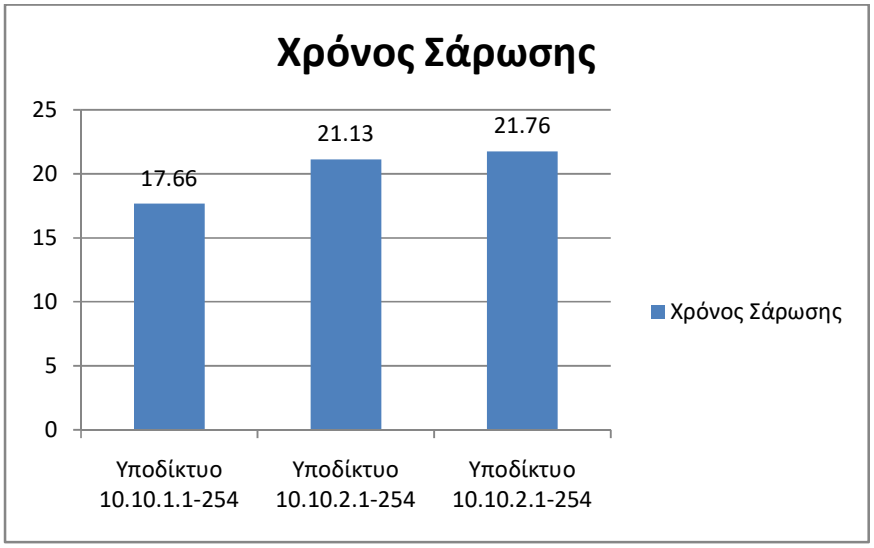
Ο συνολικός χρόνος που απαιτείται για την σάρωση όλων των υποδίκτυων είναι 59.54 δευτερόλεπτα.



Για ένα δίκτυο το οποίο περιέχει πενήντα τέσσερις υπολογιστές από τους οποίους οι έξι είναι πραγματικοί hosts, οι σαράντα οκτώ είναι honeypots και υπάρχουν τρία υποδίκτυα υπολογιστών έχουμε τα παρακάτω γραφήματα.

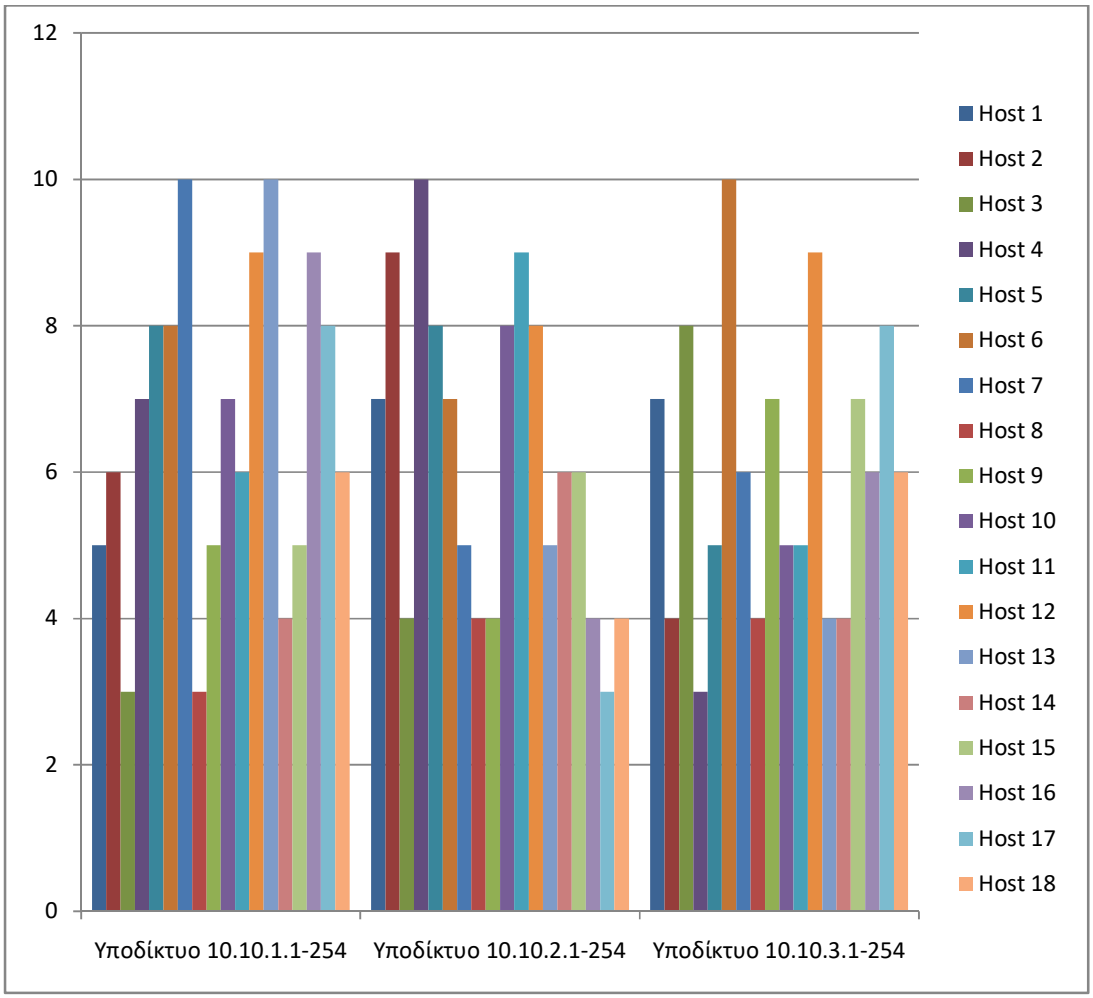


Ο συνολικός χρόνος που απαιτείται για την σάρωση όλων των υποδίκτυων είναι 60.55 δευτερόλεπτα.

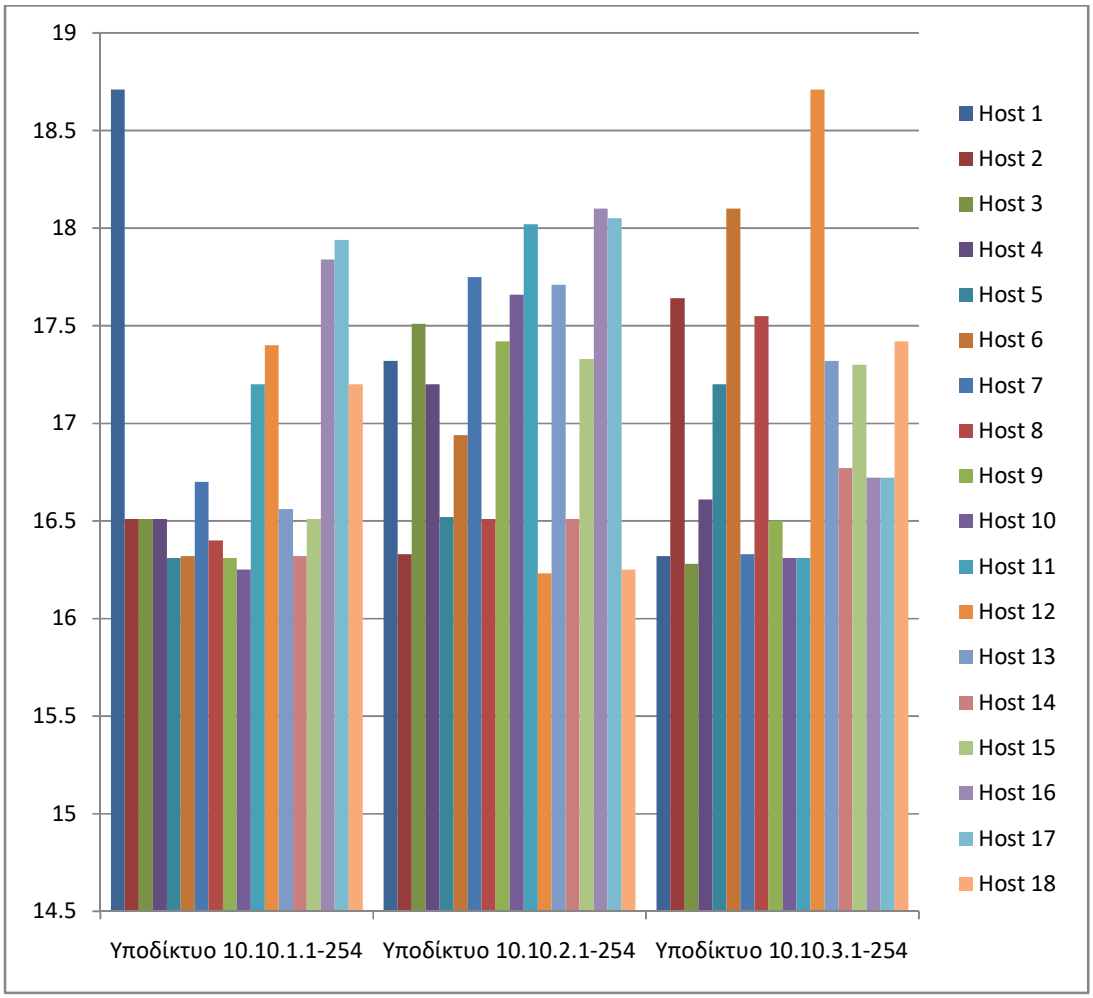


Στη συνέχεια, θα παρουσιάσουμε τα αποτελέσματα σαρώσεων που πραγματοποιήθηκαν με την χρήση του εργαλείου nmap (`εντολή nmap <ip address>`) για να εντοπίσουμε τις πόρτες που έχει ανοικτές κάθε host του δικτύου, τους οποίους εντοπίσαμε με το nmap sweep scan.

Για ένα δίκτυο το οποίο περιέχει δεκαοκτώ υπολογιστές από τους οποίους οι έξι είναι πραγματικοί hosts, οι δώδεκα είναι honeypots και υπάρχουν τρία υποδίκτυα υπολογιστών έχουμε το παρακάτω γράφημα για το πλήθος των πορτών ανά host.



Ο χρόνος σε δευτερόλεπτα που θα χρειαστεί ο επιτιθέμενος για να σαρώσει κάθε host παρουσιάζεται στο παρακάτω διάγραμμα.



Με βάση τις **πέντε κύριες ιδιότητες** [15] που πρέπει να έχει ένας αμυντικός μηχανισμός παραπλάνησης, ο συγκεκριμένος μηχανισμός αμυντικής παραπλάνησης ικανοποιεί τις παρακάτω:

Ιδιότητα 1: Να αυξάνει τον φόρτο εργασίας του εισβολέα,

Ιδιότητα 2: Να επιτρέπει στους αμυνόμενους να παρακολουθούν τις επιθέσεις και να ανταποκριθούν πριν οι αντίπαλοι πετύχουν το σκοπό τους,

Ιδιότητα 5: Να αυξάνει την αβεβαιότητα του εισβολέα.

Ικανοποιεί έως ένα βαθμό την **ιδιότητα 4**, δηλαδή να αυξάνει την πολυπλοκότητα που απαιτείται για την επίθεση, ενώ δεν ικανοποιεί την **ιδιότητα 3,** να εξαντλήσει τους πόρους του αντιπάλου.

Όσον αφορά την περαιτέρω εξέλιξη της παρούσας εργασίας θα μπορούσαμε να αναφέρουμε ότι αρχικά, θα μπορούσε να λυθεί το πρόβλημα με τη διακοπή των συνδέσεων TCP όταν αλλάζουν οι διευθύνσεις MAC και IP των υπολογιστών του δικτύου. Στη συνέχεια, θα μπορούσαμε να μεταφέρουμε τον παραπάνω μηχανισμό αμυντικής παραπλάνησης έξω από το εικονικό περιβάλλον του εξομοιωτή mininet, δηλαδή σε ένα δίκτυο το οποίο να αποτελείται από πραγματικούς υπολογιστές ή εικονικές μηχανές, switches συμβατά με την τεχνολογία SND και SDN controllers, με σκοπό τον περαιτέρω πειραματισμό με διάφορων ειδών επιθέσεις.

Παράρτημα

Στο παράρτημα παρουσιάζεται ο κώδικάς των εργαλείων που αναφέραμε στο πρακτικό μέρος της εργασίας.

Ο κώδικας του iController.py

iController.py

```
from pox.core import core
import pox.openflow.nicira as nx
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import IPAddr, EthAddr
from pox.lib.packet.arp import arp
from networkReader import NetworkReader
from createNetworkTopology import CreateNetworkTopology
import thread

log = core.getLogger()

class LearningSwitch (object):
    def __init__ (self, connection, transparent):
        self.connection = connection
        self.transparent = transparent
        connection.addListeners(self)

        self.ct = CreateNetworkTopology()
        self.ct.run()
        filename = self.ct.getFilename()
        print(filename)
        #filename = "/home/mininet/ibh/networkView.txt"
        self.net = NetworkReader(filename)
        self.net.readNetworkView()

        self.packetHandlerIP = self.net.getPacketHandlerIP()
        self.packetHandlerMAC = self.net.getPacketHandlerMAC()
        self.packetHandlerSwitchPort = int(self.net.getPacketHandlerRealSwitchPort())

        self.knownHosts = {}
        self.attacker = []
        self.appPorts_limit = 5
        self.SYN_limit = 3

        self.tcp_honeypot_ip = "10.0.0.5"
        self.tcp_honeypot_mac = "00:00:00:00:00:05"
        self.tcp_honeypot_switchPort = 5

        self.honeypots = self.net.getHoneypots()

        """
        flowRule = nx.nx_flow_mod()
        flowRule.match.of_eth_type = ethernet.IP_TYPE
        flowRule.match.of_ip_proto = ipv4.UDP_PROTOCOL
        flowRule.match.in_port = 2
        flowRule.match.eth_src = "00:00:00:00:00:02"
        flowRule.match.ip_src = "10.0.0.2"
        flowRule.match.ip_dst = "10.0.1.11"
        flowRule.match.eth_dst = "00:00:00:af:4d:07"
        flowRule.idle_timeout = 0
        #flowRule.hard_timeout = 2
        flowRule.priority = 3
        flowRule.actions.append(of.ofp_action_output(port                                =
```

```
int(self.packetHandlerSwitchPort)))
        self.connection.send(flowRule)

        flowRule = nx.nx_flow_mod()
        flowRule.match.of_eth_type = ethernet.IP_TYPE
        flowRule.match.of_ip_proto = ipv4.UDP_PROTOCOL
        flowRule.match.in_port = int(self.packetHandlerSwitchPort)
        flowRule.match.eth_src = "00:00:00:af:4d:07"
        flowRule.match.ip_src = "10.0.1.11"
        flowRule.match.ip_dst = "10.0.0.2"
        flowRule.match.eth_dst = "00:00:00:00:00:02"
        flowRule.idle_timeout = 0
        #flowRule.hard_timeout = 2
        flowRule.priority = 3
        flowRule.actions.append(of.ofp_action_output(port = 2))
        self.connection.send(flowRule)
        """
        thread.start_new_thread( self.updateTopology, (300, ) )

    def updateTopology(self, delay):
        while True:
            time.sleep(delay)
            self.ct = CreateNetworkTopology()
            self.ct.run()
            filename = self.ct.getFilename()
            print(filename)
            #filename = "/home/mininet/ibh/networkView.txt"
            self.net = NetworkReader(filename)
            self.net.readNetworkView()


    def _handle_PacketIn (self, event):
        packet = event.parsed
        in_port = event.port
        #print(packet.payload)


        if packet.type == packet.ARP_TYPE:
            #print(packet.payload)
            #print(packet.src)
            #print(packet.payload)
            #print(packet.dst)
            print(packet)


            if packet.payload.opcode == arp.REQUEST:
                #print("ARP Request")

                flowRule = nx.nx_flow_mod()
                flowRule.match.in_port = in_port
                flowRule.match.NXM_OF_ETH_TYPE = ethernet.ARP_TYPE
                flowRule.match.NXM_OF_ARP_OP = arp.REQUEST
                flowRule.match.eth_src = EthAddr(packet.src)
                flowRule.idle_timeout = 60
                flowRule.priority = 8
                flowRule.actions.append(of.ofp_action_output(port                                    =
self.packetHandlerSwitchPort))
                self.connection.send(flowRule)

                currentOutPort = self.packetHandlerSwitchPort
                flowRule = nx.nx_flow_mod()
                flowRule.match.in_port = self.packetHandlerSwitchPort
                flowRule.match.NXM_OF_ETH_TYPE = ethernet.ARP_TYPE
                flowRule.match.NXM_OF_ARP_OP = arp.REPLY
                flowRule.match.eth_dst = EthAddr(packet.src)
                flowRule.idle_timeout = 60
                flowRule.priority = 8
                flowRule.actions.append(of.ofp_action_output(port = in_port))
                self.connection.send(flowRule)

            elif packet.payload.opcode == arp.REPLY:
```

```
                print("ARP Reply ")
            else:
                print("Some other ARP opcode.")


        if packet.type == ethernet.IP_TYPE:
            print("IP TYPE")
            ip_packet = packet.payload

            src_ip = ip_packet.srcip
            dst_ip = ip_packet.dstip

            #src_mac = self.net.getHostRealMACAddressFromRealIPAddress(str(src_ip))
            #if src_mac == None:
            #                                                        src_mac             =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(src_ip))

            #dst_mac = self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(dst_ip))
            #if dst_mac == None:
            #    dst_mac = self.net.getHostRealMACAddressFromRealIPAddress(str(dst_ip))

            #dst_switchPort = self.net.getHostSwitchPortFromVirtualIPAddress(str(dst_ip))
            #if dst_switchPort == None:
            #                                                dst_switchPort            =
self.net.getHostSwitchPortFromRealIPAddress(str(dst_ip))


            #print("src_ip {}".format(src_ip))
            #print("dst_ip {}".format(dst_ip))

            #print("src_mac {}".format(src_mac))
            #print("dst_mac {}".format(dst_mac))

            #print("dst_switchPort {}".format(dst_switchPort))


            ###   ICMP   ###
            if ip_packet.protocol == ipv4.ICMP_PROTOCOL:
                print("ICMP packet")
                src_mac = self.net.getHostRealMACAddressFromRealIPAddress(str(src_ip))
                if src_mac == None:
                    src_mac                                                                 =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(src_ip))

                dst_mac                                                                     =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(dst_ip))
                if dst_mac == None:
                    dst_mac                                                                 =
self.net.getHostRealMACAddressFromRealIPAddress(str(dst_ip))

                dst_switchPort                                                              =
self.net.getHostVirtualSwitchPortFromVirtualIPAddress(str(dst_ip))
                if dst_switchPort == None:
                    dst_switchPort                                                          =
self.net.getHostVirtualSwitchPortFromRealIPAddress(str(dst_ip))
                    if dst_switchPort == None:
                        dst_switchPort = 1

                if dst_ip in self.honeypots:
                    print('ICMP interaction with honeypot {}.'.format(dst_ip))
                    if not src_ip in self.attacker:
                        self.attacker.append(src_ip)
                        print('Add {} as attacker.'.format(src_ip))

                flowRule = nx.nx_flow_mod()
                flowRule.match.of_eth_type = ethernet.IP_TYPE
                flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
                #flowRule.match.of_icmp_code = 8
                flowRule.match.of_icmp_type = 8
                flowRule.match.in_port = in_port
                flowRule.match.eth_src = src_mac
```

```
                flowRule.match.ip_src = src_ip
                flowRule.match.ip_dst = dst_ip
                flowRule.match.eth_dst = dst_mac
                flowRule.idle_timeout = 30
                flowRule.priority = 7
                flowRule.actions.append(of.ofp_action_output(port = int(dst_switchPort)))
                self.connection.send(flowRule)

                flowRule = nx.nx_flow_mod()
                flowRule.match.of_eth_type = ethernet.IP_TYPE
                flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
                #flowRule.match.of_icmp_code = 8
                flowRule.match.of_icmp_type = 0
                flowRule.match.in_port = int(dst_switchPort)
                flowRule.match.eth_src = dst_mac
                flowRule.match.ip_src = dst_ip
                flowRule.match.ip_dst = src_ip
                flowRule.match.eth_dst = src_mac
                flowRule.idle_timeout = 30
                flowRule.priority = 7
                flowRule.actions.append(of.ofp_action_output(port = in_port))
                self.connection.send(flowRule)



            ###   TCP   ###
            if ip_packet.protocol == ipv4.TCP_PROTOCOL:
                print("TCP packet")

                print("src_ip: {}".format(src_ip))
                print("dst_ip: {}".format(dst_ip))

                src_mac = self.net.getHostRealMACAddressFromRealIPAddress(str(src_ip))
                if src_mac == None:
                    src_mac                                                    =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(src_ip))
                print("src_mac: {}".format(src_mac))

                dst_mac                                                        =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(dst_ip))
                if dst_mac == None:
                    dst_mac                                                    =
self.net.getHostRealMACAddressFromRealIPAddress(str(dst_ip))
                print("dst_mac: {}".format(dst_mac))

                real_dst_switchPort                                            =
self.net.getHostRealSwitchPortFromVirtualIPAddress(str(dst_ip))
                if real_dst_switchPort == None:
                    real_dst_switchPort                                        =
self.net.getHostRealSwitchPortFromRealIPAddress(str(dst_ip))
                print("real_dst_switchPort: {}".format(real_dst_switchPort))

                virtual_dst_switchPort                                         =
self.net.getHostVirtualSwitchPortFromVirtualIPAddress(str(dst_ip))
                if virtual_dst_switchPort == None:
                    virtual_dst_switchPort                                     =
self.net.getHostVirtualSwitchPortFromRealIPAddress(str(dst_ip))
                print("virtual_dst_switchPort: {}".format(virtual_dst_switchPort))

                tcp_found = packet.find('tcp')
                if tcp_found:

                    appPort = tcp_found.dstport

                    if tcp_found.flags == 2: # SYN
                        print "SYN found"

                        if not src_ip in self.knownHosts.keys():
                            newHost = {
                                    src_ip:
                                    {
```

```
                                        "SYN_counter": 1,
                                        "appPorts": [appPort]
                                        }
                                    }
                                self.knownHosts.update(newHost)
                            else:
                                for key in self.knownHosts:
                                    self.knownHosts[key]['SYN_counter']                    =
int(self.knownHosts[key]['SYN_counter']) + 1
                                    if not appPort in self.knownHosts[key]['appPorts']:
                                        self.knownHosts[key]['appPorts'].append(appPort)
                                        if    (    len(self.knownHosts[key]['appPorts'])    >
self.appPorts_limit
                                                and
                                                int(self.knownHosts[key]['SYN_counter'])    >
self.SYN_limit ):

                                            if not src_ip in self.attacker:
                                                self.attacker.append(src_ip)
                                                print('Add {} as attacker.'.format(src_ip))

                            print("Known Hosts: {}".format(self.knownHosts))

                            """
                            if self.SYN_counter > self.SYN_limit:
                                if not src_ip in self.attacker:
                                    self.attacker.append(src_ip)
                                    print('Add {} as attacker.'.format(src_ip))
                            self.SYN_counter += 1
                            """

                        if tcp_found.flags == 18: # SYN ACK
                            print("SYN ACK found!")
                            print("Port {} is open.".format(tcp_found.srcport))

                    if src_ip in self.attacker:
                        print("Attacker: {}".format(src_ip))
                        flowRule = of.ofp_flow_mod()
                        flowRule.prioriry = 6
                        flowRule.match.dl_type = 0x800 # IPv4
                        flowRule.idle_timeout = 30
                        #flowRule.hard_timeout = 30
                        flowRule.match.nw_dst = dst_ip

flowRule.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(self.tcp_honeypot_ip)))

flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(self.tcp_honeypot_mac)))
                        flowRule.actions.append(of.ofp_action_output(port                      =
self.tcp_honeypot_switchPort))
                        #flowRule.actions.append(of.ofp_action_output(port                      =
int(virtual_dst_switchPort)))
                        self.connection.send(flowRule)

                        flowRule = of.ofp_flow_mod()
                        flowRule.priority = 6
                        flowRule.match.dl_type = 0x800
                        flowRule.idle_timeout = 30
                        #flowRule.hard_timeout = 30
                        flowRule.match.nw_dst = IPAddr(src_ip)

flowRule.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(dst_ip)))

flowRule.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(dst_mac)))

flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(src_mac)))
                        flowRule.actions.append(of.ofp_action_output(port = in_port))
                        self.connection.send(flowRule)
                    else:
                        flowRule = of.ofp_flow_mod()
                        flowRule.prioriry = 6
                        flowRule.match.dl_type = 0x800 # IPv4
```

```
                    flowRule.idle_timeout = 30
                    #flowRule.hard_timeout = 30
                    flowRule.match.nw_dst = dst_ip

flowRule.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(self.net.getHostRealIPAddres
sFromVirtualIPAddress(str(dst_ip)))))

flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(self.net.getHostRealMACAddr
essFromVirtualMACAddress(str(dst_mac)))))
                    flowRule.actions.append(of.ofp_action_output(port                    =
int(real_dst_switchPort)))
                    self.connection.send(flowRule)

                    flowRule = of.ofp_flow_mod()
                    flowRule.priority = 6
                    flowRule.match.dl_type = 0x800
                    flowRule.idle_timeout = 30
                    #flowRule.hard_timeout = 30
                    flowRule.match.nw_dst = IPAddr(src_ip)

flowRule.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(dst_ip)))

flowRule.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(dst_mac)))

flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(src_mac)))
                    flowRule.actions.append(of.ofp_action_output(port = in_port))
                    self.connection.send(flowRule)

                    """
                    flowRule = nx.nx_flow_mod()
                    flowRule.match.of_eth_type = ethernet.IP_TYPE
                    flowRule.match.of_ip_proto = ipv4.TCP_PROTOCOL
                    flowRule.match.in_port = in_port
                    flowRule.match.eth_src = src_mac
                    flowRule.match.ip_src = src_ip
                    flowRule.match.ip_dst                                            =
self.net.getHostRealIPAddressFromVirtualIPAddress(str(dst_ip))
                    flowRule.match.eth_dst                                           =
self.net.getHostRealMACAddressFromVirtualMACAddress(str(dst_mac))
                    flowRule.idle_timeout = 30
                    #flowRule.hard_timeout = 2
                    flowRule.priority = 5
                    #flowRule.actions.append(of.ofp_action_output(port                    =
int(dst_switchPort)))
                    flowRule.actions.append(of.ofp_action_output(port = 3))
                    self.connection.send(flowRule)

                    flowRule = nx.nx_flow_mod()
                    flowRule.match.of_eth_type = ethernet.IP_TYPE
                    flowRule.match.of_ip_proto = ipv4.TCP_PROTOCOL
                    flowRule.match.in_port = int(dst_switchPort)
                    flowRule.match.eth_src = dst_mac
                    flowRule.match.ip_src = dst_ip
                    flowRule.match.ip_dst = src_ip
                    flowRule.match.eth_dst = src_mac
                    flowRule.idle_timeout = 30
                    #flowRule.hard_timeout = 2
                    flowRule.priority = 5
                    flowRule.actions.append(of.ofp_action_output(port = in_port))
                    self.connection.send(flowRule)
                    """

            ###   UDP   ###
            if ip_packet.protocol == ipv4.UDP_PROTOCOL:
                udp_found = packet.find('udp')
                if udp_found.dstport == 53:
                    print("UDP packet   DNS")
                    src_mac                                                          =
self.net.getHostRealMACAddressFromRealIPAddress(str(src_ip))
                    if src_mac == None:
                        src_mac                                                      =
```

```
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(src_ip))

                    dst_mac                                                                 =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(dst_ip))
                    if dst_mac == None:
                        dst_mac                                                             =
self.net.getHostRealMACAddressFromRealIPAddress(str(dst_ip))

                    dst_switchPort                                                          =
self.net.getHostVirtualSwitchPortFromVirtualIPAddress(str(dst_ip))
                    if dst_switchPort == None:
                        dst_switchPort                                                      =
self.net.getHostVirtualSwitchPortFromRealIPAddress(str(dst_ip))

                    flowRule = of.ofp_flow_mod()
                    flowRule.prioriry = 5
                    flowRule.match.dl_type = 0x800 # IPv4
                    flowRule.idle_timeout = 30
                    #flowRule.hard_timeout = 0
                    flowRule.match.nw_dst = dst_ip

flowRule.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(self.packetHandlerIP)))

flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(self.packetHandlerMAC)))
                    flowRule.actions.append(of.ofp_action_output(port                       =
int(self.packetHandlerSwitchPort)))
                    self.connection.send(flowRule)

                    flowRule = of.ofp_flow_mod()
                    flowRule.priority = 5
                    flowRule.match.dl_type = 0x800
                    flowRule.idle_timeout = 30
                    #flowRule.hard_timeout = 0
                    flowRule.match.nw_dst = IPAddr(src_ip)

flowRule.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(dst_ip)))

flowRule.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(dst_mac)))

flowRule.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(src_mac)))
                    flowRule.actions.append(of.ofp_action_output(port = in_port))
                    self.connection.send(flowRule)

                elif udp_found.dstport >= 33434 and udp_found.dstport <= 33523:
                    print("UDP packet   Traceroute")

                    src_mac                                                                 =
self.net.getHostRealMACAddressFromRealIPAddress(str(src_ip))
                    if src_mac == None:
                        src_mac                                                             =
self.net.getHostVirtualMACAddressFromVirtualIPAddress(str(src_ip))


                    dst_switchPort                                                          =
self.net.getHostVirtualSwitchPortFromVirtualIPAddress(str(dst_ip))
                    if dst_switchPort == None:
                        dst_switchPort                                                      =
self.net.getHostVirtualSwitchPortFromRealIPAddress(str(dst_ip))

                    if dst_ip in self.honeypots:
                        print('Traceroute interaction with honeypot {}.'.format(dst_ip))
                        if not src_ip in self.attacker:
                            self.attacker.append(src_ip)
                            print('Add {} as attacker.'.format(src_ip))

                    flowRule = nx.nx_flow_mod()
                    flowRule.match.of_eth_type = ethernet.IP_TYPE
                    flowRule.match.of_ip_proto = ipv4.UDP_PROTOCOL
                    flowRule.match.in_port = in_port
                    #flowRule.match.eth_src = src_mac
                    #flowRule.match.ip_src = src_ip
```

```
                    #flowRule.match.ip_dst = dst_ip
                    #flowRule.match.eth_dst = dst_mac
                    flowRule.idle_timeout = 60
                    #flowRule.hard_timeout = 0
                    flowRule.priority = 4
                    flowRule.actions.append(of.ofp_action_output(port                    =
int(dst_switchPort)))
                    self.connection.send(flowRule)

                    flowRule = nx.nx_flow_mod()
                    flowRule.match.of_eth_type = ethernet.IP_TYPE
                    flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
                    flowRule.match.of_icmp_type = 11
                    flowRule.match.in_port = int(dst_switchPort)
                    #flowRule.match.eth_src = dst_mac
                    #flowRule.match.ip_src = dst_ip
                    #flowRule.match.ip_dst = src_ip
                    #flowRule.match.eth_dst = src_mac
                    flowRule.idle_timeout = 60
                    #flowRule.hard_timeout = 0
                    flowRule.priority = 4
                    flowRule.actions.append(of.ofp_action_output(port = in_port))
                    self.connection.send(flowRule)

                    flowRule = nx.nx_flow_mod()
                    flowRule.match.of_eth_type = ethernet.IP_TYPE
                    flowRule.match.of_ip_proto = ipv4.ICMP_PROTOCOL
                    flowRule.match.of_icmp_type = 3
                    flowRule.match.in_port = int(dst_switchPort)
                    #flowRule.match.eth_src = dst_mac
                    #flowRule.match.ip_src = dst_ip
                    #flowRule.match.ip_dst = src_ip
                    #flowRule.match.eth_dst = src_mac
                    flowRule.idle_timeout = 0
                    flowRule.hard_timeout = 0
                    flowRule.priority = 4
                    flowRule.actions.append(of.ofp_action_output(port = in_port))
                    self.connection.send(flowRule)



class ilearn (object):
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False):
  core.registerNew(ilearn, str_to_bool(transparent))
```

Ο κώδικας του networkReader.py

networkReader.py

```
import pprint
class NetworkReader(object):

    def __init__(self, filename):
        self.filename = filename
        self.packetHandler = {}
        self.hosts = {}
        self.fakerouters = {}
        self.routes = {}

    def readNetworkView(self):
        host_counter = 0
        fakerouter_counter = 0
        route_counter = 0

        with open(self.filename) as file:
            lines = file.readlines()
            for line in lines:
                line = line.replace("\n","")
                args = line.split(",")
                if args[0] == "packetHandler":
                    handler = {
                            "shortName": args[1],
                            "realIPAddress": args[2],
                            "realMACAddress": args[3],
                            "virtualIPAddress": args[4],
                            "virtualMACAddress": args[5],
                            "virtualSwitchPort": args[6],
                            "realSwitchPort": args[7]
                            }
                    self.packetHandler = handler
                if args[0] == "host":
                    host_counter += 1
                    host = {
                            host_counter:
                            {
                                "shortName": args[1],
                                "realIPAddress": args[2],
                                "realMACAddress": args[3],
                                "virtualIPAddress": args[4],
                                "virtualMACAddress": args[5],
                                "virtualSwitchPort": args[6],
                                "realSwitchPort": args[7]

                                }
                            }
                    self.hosts.update(host)
                if args[0] == "fakerouter":
                    fakerouter_counter += 1
                    fakerouter = {
                            fakerouter_counter:
                            {
                                "shortName": args[1],
                                "virtualIPAddress": args[2],
                                "virtualMACAddress": args[3],
                                "switchPort": args[4]
                                }
                            }
                    self.fakerouters.update(fakerouter)
                if args[0] == "route":
                    route_counter += 1
                    hops_length = len(args) - 3 # len(args) - <route> - <source> -
<destination>
                    hops = []
                    if hops_length == 1:
                        hop1 = args[2]
```

```
                        hops.append(self.getHostRealIPAddressFromName(args[1]))
                        hops.append(self.getFakerouterVirtualIPAddressFromName(hop1))
                        hops.append(self.getHostVirtualIPAddressFromName(args[len(args) -
1]))
                    elif hops_length == 2:
                        hop1 = args[2]
                        hop2 = args[3]
                        hops.append(self.getHostRealIPAddressFromName(args[1]))
                        hops.append(self.getFakerouterVirtualIPAddressFromName(hop1))
                        hops.append(self.getFakerouterVirtualIPAddressFromName(hop2))
                        hops.append(self.getHostVirtualIPAddressFromName(args[len(args) -
1]))
                    elif hops_length == 3:
                        hop1 = args[2]
                        hop2 = args[3]
                        hop3 = args[4]
                        hops.append(self.getHostRealIPAddressFromName(args[1]))
                        hops.append(self.getFakerouterVirtualIPAddressFromName(hop1))
                        hops.append(self.getFakerouterVirtualIPAddressFromName(hop2))
                        hops.append(self.getFakerouterVirtualIPAddressFromName(hop3))
                        hops.append(self.getHostVirtualIPAddressFromName(args[len(args) -
1]))

                    route = {
                            route_counter:
                            {
                                "source": self.getHostRealIPAddressFromName(args[1]),
                                "destination":
self.getHostVirtualIPAddressFromName(args[len(args) - 1]),
                                "hops": hops
                                }
                            }
                    self.routes.update(route)

    ### GENERAL INFO ###
    def getPacketHandler(self):
        return self.packetHandler

    def getHosts(self):
        return self.hosts

    def getFakerouters(self):
        return self.fakerouters

    def getRoutes(self):
        return self.routes
    ### END OF GENERAL INFO ###

    ### PACKET HANDLER ###

    def getPacketHandlerIP(self):
        return self.packetHandler['realIPAddress']

    def getPacketHandlerMAC(self):
        return self.packetHandler['realMACAddress']

    def getPacketHandlerRealSwitchPort(self):
        return self.packetHandler['realSwitchPort']

    ### END OF PACKET HANDLER ###

    ### HOST INFO ###

    def getHostVirtualIPAddressFromName(self, name):
        for key in self.hosts:
            if self.hosts[key]['shortName'] == name:
                return self.hosts[key]['virtualIPAddress']

    def getHostRealIPAddressFromName(self, name):
        for key in self.hosts:
            if self.hosts[key]['shortName'] == name:
```

```
            return self.hosts[key]['realIPAddress']

    def getHostRealMACAddressFromName(self, name):
        for key in self.hosts:
            if self.hosts[key]['shortName'] == name:
                return self.hosts[key]['realMACAddress']

    def getHostVirtualMACAddressFromName(self, name):
        for key in self.hosts:
            if self.hosts[key]['shortName'] == name:
                return self.hosts[key]['virtualMACAddress']

    def getHostVirtualMACAddressFromVirtualIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['virtualIPAddress'] == ipaddr:
                return self.hosts[key]['virtualMACAddress']

    def getHostVirtualMACAddressFromRealIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['realIPAddress'] == ipaddr:
                return self.hosts[key]['virtualMACAddress']

    def getHostRealMACAddressFromRealIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['realIPAddress'] == ipaddr:
                return self.hosts[key]['realMACAddress']

    def getHostRealMACAddressFromVirtualMACAddress(self, macaddr):
        for key in self.hosts:
            if self.hosts[key]['virtualMACAddress'] == macaddr:
                return self.hosts[key]['realMACAddress']

    def getHostRealIPAddressFromVirtualIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['virtualIPAddress'] == ipaddr:
                return self.hosts[key]['realIPAddress']

    def getHostVirtualSwitchPortFromVirtualIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['virtualIPAddress'] == ipaddr:
                return self.hosts[key]['virtualSwitchPort']

    def getHostRealSwitchPortFromVirtualIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['virtualIPAddress'] == ipaddr:
                return self.hosts[key]['realSwitchPort']

    def getHostRealSwitchPortFromRealIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['realIPAddress'] == ipaddr:
                return self.hosts[key]['realSwitchPort']

    def getHostVirtualSwitchPortFromRealIPAddress(self, ipaddr):
        for key in self.hosts:
            if self.hosts[key]['realIPAddress'] == ipaddr:
                return self.hosts[key]['virtualSwitchPort']

    def getHoneypots(self):
        honeypotsList = []
        for key in self.hosts:
            if self.hosts[key]['shortName'][0:2] == 'hp':
                honeypotsList.append(self.hosts[key]['virtualIPAddress'])
        return honeypotsList


    ### END OF HOST INFO ###

    ### FAKEROUTER INFO ###
    def getFakerouterVirtualIPAddressFromName(self, name):
        for key in self.fakerouters:
            if self.fakerouters[key]['shortName'] == name:
```

```
                return self.fakerouters[key]['virtualIPAddress']

    def getFakerouterVirtualMACAddressFromName(self, name):
        for key in self.fakerouters:
            if self.fakerouters[key]['shortName'] == name:
                return self.fakerouters[key]['virtualMACAddress']

    def getFakerouterVirtualMACAddressFromIP(self, ipaddr):
        for key in self.fakerouters:
            if self.fakerouters[key]['virtualIPAddress'] == ipaddr:
                return self.fakerouters[key]['virtualMACAddress']

    def getRoute(self, source, destination):
        for key in self.routes:
            if self.routes[key]['source'] == source and self.routes[key]['destination']
== destination:
                return self.routes[key]['hops']

    ### END OF FAKEROUTER INFO ###
```

Ο κώδικας του createNetworkTopology.py

createNetworkTopology.py

```
from random import sample
from random import randint
from pprint import pprint

class CreateNetworkTopology(object):

    def __init__(self):
        self.virtualIPAddressList = []
        self.virtualMACAddressList = []
        self.hostsNum = 6
        self.honeypotsNum = 9
        self.subnetsNum = 3
        self.content = []
        self.filename = '/home/mininet/ibh/networkView.txt'

    def packetHandler(self):
        shortName = 'h1'
        realIPAddress = '10.0.0.1'
        realMACAddress = '00:00:00:00:00:01'


        while True:
            tempIP = self.createVirtualIPAddress('10.10.1.')
            if not tempIP in self.virtualIPAddressList:
                self.virtualIPAddressList.append(tempIP)
                virtualIPAddress = tempIP
                break

        while True:
            tempMAC = self.createVirtualMACAddress()
            if not tempMAC in self.virtualMACAddressList:
                self.virtualMACAddressList.append(tempMAC)
                virtualMACAddress = tempMAC
                break

        virtualSwitchPort = '1'
        realSwitchPort = '1'
        row = 'packetHandler' + ',' + shortName + ',' + realIPAddress + ',' +
realMACAddress + ',' + virtualIPAddress + ',' + virtualMACAddress + ',' +
virtualSwitchPort + ',' + realSwitchPort

        self.content.append(row)

    def host(self):
```

```
        prefix = ''
        hostsPerNetwork = self.hostsNum / self.subnetsNum
        networkUpLimit = hostsPerNetwork
        networkDownLimit = 0
        networkNumber = 1

        for i in range(2, self.hostsNum + 1):
            shortName = 'h' + str(i)
            realIPAddress = '10.0.0.' + str(i)
            if i < 15:
                realMACAddress = '00:00:00:00:00:0' + str(hex(i).split('x')[-1])
            else:
                realMACAddress = '00:00:00:00:00:' + str(hex(i).split('x')[-1])

            if i <= networkUpLimit + hostsPerNetwork and i > networkDownLimit +
hostsPerNetwork:
                networkUpLimit += hostsPerNetwork
                networkDownLimit += hostsPerNetwork
                networkNumber += 1
            if i <= networkUpLimit and i > networkDownLimit:
                prefix = '10.10.' + str(networkNumber) + '.'

            #print("i {}, networkUpLimit {}, networkDownLimit {}, networkNumber
{}".format(i, networkUpLimit, networkDownLimit, networkNumber))


            while True:
                tempIP = self.createVirtualIPAddress(prefix)
                if not tempIP in self.virtualIPAddressList:
                    self.virtualIPAddressList.append(tempIP)
                    virtualIPAddress = tempIP
                    break

            while True:
                tempMAC = self.createVirtualMACAddress()
                if not tempMAC in self.virtualMACAddressList:
                    self.virtualMACAddressList.append(tempMAC)
                    virtualMACAddress = tempMAC
                    break

            virtualSwitchPort = '1'
            realSwitchPort = str(i)
            row = 'host' + ',' +  shortName + ',' + realIPAddress + ',' + realMACAddress
+ ',' + virtualIPAddress + ',' + virtualMACAddress + ',' + virtualSwitchPort + ',' +
realSwitchPort

            self.content.append(row)

    def honeypot(self):
        prefix = ''
        honeypotsPerNetwork = self.honeypotsNum / self.subnetsNum
        networkUpLimit = honeypotsPerNetwork
        networkDownLimit = 0
        networkNumber = 1

        for i in range(1, self.honeypotsNum + 1):
            shortName = 'hp' + str(i)
            realIPAddress = '10.0.0.5'
            realMACAddress = '00:00:00:00:00:05'

            if i <= networkUpLimit + honeypotsPerNetwork and i > networkDownLimit +
honeypotsPerNetwork:
                networkUpLimit += honeypotsPerNetwork
                networkDownLimit += honeypotsPerNetwork
                networkNumber += 1
            if i <= networkUpLimit and i > networkDownLimit:
                prefix = '10.10.' + str(networkNumber) + '.'

            #print("i {}, networkUpLimit {}, networkDownLimit {}, networkNumber
{}".format(i, networkUpLimit, networkDownLimit, networkNumber))
```

```
            while True:
                tempIP = self.createVirtualIPAddress(prefix)
                if not tempIP in self.virtualIPAddressList:
                    self.virtualIPAddressList.append(tempIP)
                    virtualIPAddress = tempIP
                    break

            while True:
                tempMAC = self.createVirtualMACAddress()
                if not tempMAC in self.virtualMACAddressList:
                    self.virtualMACAddressList.append(tempMAC)
                    virtualMACAddress = tempMAC
                    break

            virtualSwitchPort = '1'
            realSwitchPort = '5'
            row = 'host' + ',' +  shortName + ',' + realIPAddress + ',' + realMACAddress
+ ',' + virtualIPAddress + ',' + virtualMACAddress + ',' + virtualSwitchPort + ',' +
realSwitchPort

            self.content.append(row)

    def fakerouter(self):
        self.content.append('fakerouter,fr1_if0,10.10.1.1,00:00:00:2b:65:14,1')
        self.content.append('fakerouter,fr1_if1,10.10.13.1,00:00:00:43:1d:7a,1')
        self.content.append('fakerouter,fr2_if0,10.10.13.2,00:00:00:93:82:cd,1')
        self.content.append('fakerouter,fr2_if1,10.10.14.1,00:00:00:83:de:7f,1')
        self.content.append('fakerouter,fr2_if2,10.10.2.1,00:00:00:c7:5e:02,1')
        self.content.append('fakerouter,fr3_if0,10.10.3.1,00:00:00:d7:8b:03,1')
        self.content.append('fakerouter,fr3_if1,10.10.14.2,00:00:00:c2:49:93,1')

    def route(self):
        self.content.append('route,h2,hp1')
        self.content.append('route,h2,hp2')
        self.content.append('route,h2,hp3')
        self.content.append('route,h2,fr1_if0,fr2_if0,h3')
        self.content.append('route,h2,fr1_if0,fr2_if0,h4')
        self.content.append('route,h2,fr1_if0,fr2_if0,hp4')
        self.content.append('route,h2,fr1_if0,fr2_if0,hp5')
        self.content.append('route,h2,fr1_if0,fr2_if0,hp6')
        self.content.append('route,h2,fr1_if0,fr2_if0,fr3_if1,h5')
        self.content.append('route,h2,fr1_if0,fr2_if0,fr3_if1,h6')
        self.content.append('route,h2,fr1_if0,fr2_if0,fr3_if1,hp7')
        self.content.append('route,h2,fr1_if0,fr2_if0,fr3_if1,hp8')
        self.content.append('route,h2,fr1_if0,fr2_if0,fr3_if1,hp9')


    def createVirtualIPAddress(self, prefix):
        return prefix + str(randint(1, 254))


    def createVirtualMACAddress(self):
        mac = ""
        part1to4 = "00:02:a0:00"
        part5 = ''.join(sample('0123456789abcdef',2))
        part6 = ''.join(sample('0123456789abcdef',2))

        mac = part1to4 + ":" +  part5 + ":" + part6
        return mac

    def getFilename(self):
        return self.filename

    def run(self):
        self.packetHandler()
        self.host()
        self.honeypot()
        self.fakerouter()
        self.route()
        pprint(self.content)
```

```
        with open(self.filename, 'w') as f:
            for row in self.content:
                f.write("%s\n" % row)


if __name__ == "__main__":
    a = CreateNetworkTopology()
    a.run()
```

Ο κώδικας του iPacketHandler.py

iPacketHandler.py

```
from scapy.all import *
from scapy.layers.inet import *
from scapy.layers.dhcp import *
from scapy.layers.dns import *
from scapy.layers.inet import IP
from scapy.layers.inet import ICMP
from scapy.layers.inet import TCP
from networkReader import NetworkReader
from random import sample
import time
import thread

class PacketHandler(object):
    def __init__(self, filename):
        self.knownPackets = []
        self.filename = filename
        self.net = NetworkReader(self.filename)
        self.net.readNetworkView()

        thread.start_new_thread( self.updateTopology, (300, ) )

    def updateTopology(self, delay):
        while True:
            time.sleep(delay)
            filename = "/home/mininet/ibh/networkView.txt"
            self.net = NetworkReader(filename)
            self.net.readNetworkView()
            print("Update topology.")


    def readPackets(self, pkt):
        #print(pkt.show())
        if pkt[0] not in self.knownPackets:
            if pkt[0].haslayer(ARP):
                print("ARP packet received...")
                self.createARPResponse(pkt)

            if pkt[0].haslayer(ICMP):
                #print(pkt.show())
                print("ICMP packet received...")
                self.createICMPResponse(pkt)

            if pkt[0].haslayer(UDP):
                print("UDP packet received...")
                if pkt[0][UDP].dport == 53:
                    print("DNS packet received... ")
                    self.handleDNSPacket(pkt)

            if pkt[0].haslayer(IP) and pkt[0][IP].ttl<25:
                print("Send traceroute response...")
                self.createRouteResponse(pkt)
                #seenPkts.append(respPkt)
                #sendp(respPkt, inter=0.001, verbose=0)
```

```
    def createRouteResponse(self, pkt):
        src_eth = pkt[0][Ether].src
        dst_eth = pkt[0][Ether].dst
        src_ip = pkt[0][IP].src
        dst_ip = pkt[0][IP].dst
        cur_ttl = pkt[0][IP].ttl

        route = self.net.getRoute(str(src_ip), str(dst_ip))
        if cur_ttl <= len(route) - 1:

            hop_ip = route[cur_ttl]
            #if cur_ttl == hop and hop_ip != dst_ip:
            if hop_ip != dst_ip:
                hop_eth = self.net.getFakerouterVirtualMACAddressFromIP(hop_ip)
                #icmp time exceed during transmission
                ether = Ether( src = hop_eth, dst = src_eth)
                ip = IP(src = hop_ip, dst = src_ip)
                icmp = ICMP(type = 11, code = 0)
                #resp = ether/ip/icmp/pkt[0][IP]/pkt[0][UDP]
                resp = ether/ip/icmp/IPerror(str(pkt[0][IP]))
                #recalculate checksum
                resp.show2()
            #if cur_ttl == hop and hop_ip == dst_ip:
                # if cur_ttl>=(len(route.hops)-1) and hop_ip==dst_ip:
                #icmp destination and port unreachable
            if hop_ip == dst_ip:
                hop_eth = self.net.getHostVirtualMACAddressFromVirtualIPAddress(hop_ip)
                ether = Ether(src = hop_eth, dst = src_eth)
                ip = IP(src = hop_ip, dst = src_ip)
                icmp = ICMP(type=3, code=3)
                resp = ether/ip/icmp/IPerror(str(pkt[0][IP]))
                #recalculate checksum
                resp.show2()

            self.knownPackets.append(resp)
            sendp(resp, inter=0.001, verbose=0)


    def handleDNSPacket(self, pkt):
        #print(pkt.show())
        dp=pkt[0][IP][UDP].dport
        sp=pkt[0][IP][UDP].sport

        ip_src = pkt[0][IP].src
        ip_dst = pkt[0][IP].dst

        hw_src = pkt[0][Ether].src
        hw_dst = pkt[0][Ether].dst

        q = pkt[0][IP][DNS][DNSQR].qname
        qn = q.split('.')
        if len(qn) == 2:    # DNS lookup
            print("dns lookup {}".format(qn[0]))
            ipaddr = self.net.getHostVirtualIPAddressFromName(str(qn[0]))
        elif len(qn) >= 6:  # reverse DNS lookup
            ip = qn[3] + '.' + qn[2] + '.' + qn[1] + '.' +qn[0]
            print("reverse DNS lookup {}".format(ip))
            ipaddr = ip
            return

        if ipaddr == None:
            return

        # ipaddr = self.net.getHostVirtualIPAddressFromName('h3')

        #print("ip_src {}".format(ip_src))
        #print("ip_dst {}".format(ip_dst))
        #print("hw_src {}".format(hw_src))
        #print("hw_dst {}".format(hw_dst))
        #print("ipaddr {}".format(ipaddr))
```

```
        ether = Ether(src = hw_dst, dst = hw_src)
        ip = IP(dst = ip_src, src = ip_dst)
        udp = UDP(sport = dp, dport = sp)
        dns = DNS(
                qr = 1,
                id = pkt[0][IP][DNS].id,
                qd = DNSQR(qname=pkt[0][IP][DNS][DNSQR].qname),
                an = (
                    DNSRR(rrname=pkt[0][IP][DNS][DNSQR].qname,
                    #rdata = "10.0.0.12",
                    rdata = ipaddr,
                    ttl = 3600,
                    rclass = "IN",
                    type = "A")
                    ),
                    ar = 1) #DNSRROPT(rclass=3000)

        dns_resp = ether/ip/udp/dns

        #resp                                                                    =
IP(dst="8.8.8.8")/UDP(sport=pkt[UDP].dport)/DNS(rd=1,qr=1,ra=1,id=pkt[0][IP][DNS].id,qd=D
NSQR(qname=pkt[0][IP][DNS][DNSQR].qname))

        dns_resp.show2()
        self.knownPackets.append(dns_resp)
        sendp(dns_resp, inter=0.001, verbose=0)



    def createARPResponse(self, pkt):
        if pkt[0].op == 1: # Request
            hw_src = pkt[0].hwsrc
            ip_src = pkt[0].psrc

            ip_dst = pkt[0].pdst
            hw_dst = self.net.getHostVirtualMACAddressFromVirtualIPAddress(ip_dst)
            if hw_dst == None:
                #hw_dst = self.getRandomMACAddress()
                hw_dst = self.net.getHostVirtualMACAddressFromRealIPAddress(ip_dst)

            print("**************************")
            print("ip_src: {}".format(ip_src))
            print("hw_src: {}".format(hw_src))
            print("ip_dst: {}".format(ip_dst))
            print("hw_dst: {}".format(hw_dst))
            print("**************************")

            if not hw_dst == None:
                arp = eval(pkt[0].command())
                arp[Ether].dst = hw_src
                arp[Ether].src = hw_dst
                arp[ARP].hwdst = hw_src
                arp[ARP].hwsrc = hw_dst
                arp[ARP].pdst = ip_src
                arp[ARP].psrc = ip_dst
                arp[ARP].op = 2

                self.knownPackets.append(arp)
                sendp(arp, inter = 0.001, verbose = 0)

    def getRandomMACAddress(self):
        mac = ""
        part1to3 = "00:03:00"
        part4 = ''.join(sample('0123456789abcdef',2))
        part5 = ''.join(sample('0123456789abcdef',2))
        part6 = ''.join(sample('0123456789abcdef',2))

        mac = part1to3 + ":" + part4 + ':' +  part5 + ":" + part6
        return mac
```

```
    def createICMPResponse(self, pkt):
        if pkt[0][ICMP].type == 8:
            hw_src = pkt[0][Ether].src
            hw_dst = pkt[0][Ether].dst
            ip_src = pkt[0][IP].src
            ip_dst = pkt[0][IP].dst

            icmp = pkt[0]
            icmp[ICMP].type = 0
            icmp[ICMP].code = 0
            icmp[IP].src = ip_dst
            icmp[IP].dst = ip_src
            icmp[Ether].src = hw_dst
            icmp[Ether].dst = hw_src

            del icmp["ICMP"].chksum

            icmp.show2()
            self.knownPackets.append(icmp)
            sendp(icmp, inter = 0.0001, verbose = 0)

    def run(self):
        sniff(prn = self.readPackets)


if __name__ == "__main__":
    print("[*] iPacketHandler is running !!!")
    filename = "/home/mininet/ibh/networkView.txt"
    ph = PacketHandler(filename)
    ph.run()
```

Ο κώδικας του honeypot.py

honeypot.py

```
#!/usr/bin/env python

import sys
import time
import socket

def writeLog(client, data=''):
        separator = '='*50
        fopen = open('./honey.mmh', 'a')
        fopen.write('Time:   %s\nIP:   %s\nPort:   %d\nData:   %s\n%s\n\n'%(time.ctime(),
client[0], client[1], data, separator))
        fopen.close()

def main(host, port, motd):
        print('Starting honeypot {} {} {}'.format(host, port, motd))
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind((host, port))
        s.listen(100)
        while True:
                (insock, address) = s.accept()
                print 'Connection from: %s:%d' % (address[0], address[1])
                try:
                        insock.send('%s\n'%(motd))
                        data = insock.recv(1024)
                        insock.close()
                except socket.error, e:
                        writeLog(address)
                else:
                        writeLog(address, data)

if __name__=='__main__':
        try:
                host = sys.argv[1]
```

```
                port = int(sys.argv[2])
                motd = sys.argv[3]

                #print("host {}".format(host))
                #print("port {}".format(port))
                #print("motd {}".format(motd))

                main(host, port, motd)
        except KeyboardInterrupt:
                print 'Bye!'
                exit(0)
        except BaseException, e:
                print 'Error: %s' % (e)
                exit(1)
```

Ο κώδικας του honeypot.sh

honeypot.sh

```
#!/bin/bash

# Top 20 Ports
#    21: ftp
#    22: ssh
#    23: telnet
#    25: smtp
#    53: domain name system
#    80: http
#    110: pop3
#    111: rpcbind
#    135: msrpc
#    139: netbios-ssn
#    143: imap
#    443: https
#    445: microsoft-ds
#    993: imaps
#    995: pop3s
#    1723: pptp
#    3306: mysql
#    3389: ms-wbt-server
#    5900: vnc
#    8080: http-proxy


ip_address='10.0.0.5'
ports=(21 22 23 25 53 80 110 111 135 139 143 443 445 993 995 1723 3306 3389 5900 8080)

while true; do

        num=$((3 + RANDOM % 12))

        for i in $(seq 1 $num); do
                port=$(printf "%s\n" ${ports[@]} | shuf | head -1);
                #echo $port;

                case $port in
                        21)
                                python honeypot.py $ip_address 21 vsftpd &
                                ;;
                        23)
                                python honeypot.py $ip_address 23 telnet &
                                ;;
                        25)
                                python honeypot.py $ip_address 25 smtp &
                                ;;
                        53)
                                python honeypot.py $ip_address 53 dns &
                                ;;
```

```
                        80)
                                python honeypot.py $ip_address 80 http &
                                ;;
                        110)
                                python honeypot.py $ip_address 110 pop3 &
                                ;;
                        111)
                                python honeypot.py $ip_address 111 rpcbind &
                                ;;
                        135)
                                python honeypot.py $ip_address 135 msrpc &
                                ;;
                        139)
                                python honeypot.py $ip_address 139 netbios-ssn &
                                ;;
                        143)
                                python honeypot.py $ip_address 143 imap &
                                ;;
                        443)
                                python honeypot.py $ip_address 445 microsoft-ds &
                                ;;
                        445)
                                python honeypot.py $ip_address 445 microsoft-ds &
                                ;;
                        993)
                                python honeypot.py $ip_address 993 imaps &
                                ;;
                        995)
                                python honeypot.py $ip_address 995 pop3s &
                                ;;
                        1723)
                                python honeypot.py $ip_address 1723 pptp &
                                ;;
                        3306)
                                python honeypot.py $ip_address 3306 mysql &
                                ;;
                        3389)
                                python honeypot.py $ip_address 3389 ms-wbt-server &
                                ;;
                        5900)
                                python honeypot.py $ip_address 5900 vnc &
                                ;;
                        8080)
                                python honeypot.py $ip_address 8080 http-proxy &
                                ;;
                        *)
                                echo "Port not found!"
                                ;;
                esac
        done

        sleep 30
        # kill open ports
        pkill -f honeypot.py
done
```

Αναφορές

[1] Defensive Deception Techniques on Software Defined Network: Survey
[2] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Verı ´ssimo, Christian Esteve Rothenberg, Siamak Azodolmolky and Steve Uhlig,Member IEEE: Software-Defined Networking: A Comprehensive Survey, January 2015.
[3] Kreutz, Diego; Paulo Esteves Verı´ssimo; Siamak Azodolmolky, Software-Defined Networking: A Comprehensive Survey, 2015
[4] Layne, Conrad, CYBER ATTACKS AGAINST CRITICAL INFRASTRUCTURE, 2017
[5] Wenda, D.; Ning, D, A honeypot detection method based on characteristic analysis and environment detection, 2012
[6] Schneider, Omer; GILLER, Nir, US10015188B2, 2018
[7] HOFFMAN, KEVIN; ZAGE, DAVID; NITA-ROTARU, CRISTINA, A Survey of Attack and Defense Techniques for Reputation Systems, ACM Computing Surveys, 2009
[8] Jajodia, Sushil; K. Ghosh, Anup; Swarup, Vipin ; Wang, Cliff ; Wang, X. Sean, Moving Target Defense, 2011
[9] Lei, Cheng; Zhang,, Hong-Qi ; Tan, Jing-Lei; Zhang, Yu-Chen; Liu, Xiao-Hu, Moving Target Defense Techniques: A Survey, Security and Communication Networks, 2018
[10] Ward, B.C.; Gomez, S.R.; Skowyra, R.W.; Bigelow, D. ; Martin, J.N. ; Landry, J.W. ; Okhravi, H., Survey of Cyber Moving Targets, Lincoln Laboratory MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2018
[11] Rowshanrad, Shiva; Namvarasl, Sahar; Abdi, Vajihe; Hajizadeh, Maryam; Keshtgary, Manijeh, A survey on SDN, the future of networking, Journal of Advanced Computer Science & Technology, 2014
[12] Yanbiao Li, Dafang Zhang, Javid Taheri, and Keqin Li, SDN components and OpenFlow, 2018, available online https://pdfs.semanticscholar.org/815c/4901c8e04141a54efdec61da6e8df3518895.pdf
[13] Mininet Overview, available online http://mininet.org/overview/
[14] https://www.forcepoint.com/cyber-edu/deception-technology
[15] Cohen, Fred, The Use of Deception Techniques: Honeypots and Decoys, 2004
[16] Fraunholz, Daniel; Anton, Simon Duque; Lipps, Christoph; Reti, Daniel; Krohmer, Daniel; Pohl, Frederic; Tammen, Matthias; Schotten, Hans Dieter, Demystifying Deception Technology: A Survey, 2018
[17] HAN, XIAO, KHEIR, NIZAR, BALZAROTTI, DAVIDE, Deception Techniques in Computer Security: A Research Perspective, 2018
[18] CAI, Gui-lin; WANG, Bao-sheng, HU, Wei, WANG, Tian-zuo, Moving target defense: state of the art and characteristics, 2016
[19] Han, Wonkyu; Zhao,, Ziming; Doupé, Adam; Ahn, Gail-Joon, HoneyMix: Toward SDN-based Intelligent Honeynet, 2016
[20] Liston, Tom, LaBrea: Sticky honeypot and IDS, 2001, available at http://labrea.sourceforge.net/labrea-info.html
[21] Borders, Kevin; Falk , Laura ; Prakash, Atul, OpenFire: Using deception to reduce network attacks, Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007
[22] Shing, Leslie, An improved tarpit for network deception Master's thesis, 2016
[23] Malécot, Erwan Le, MitiBox: camouflage and deception for network scan mitigation, 2009
[24] Trassare, Samuel T., A technique for presenting a deceptive dynamic network topology Master's Thesis, 2013
[25] Smart, Matthew; Malan, G. Robert; Jahanian, Farnam, Defeating TCP/IP stack fingerprinting, Usenix Security Symposium, 2000

[26] Bowen, Brian M.; Kemerlis, Vasileios P. ; Prabhu, Pratap ; Keromytis, Angelos D. ; Stolfo, Salvatore J., Automating the injection of believable decoys to detect snooping, 3rd ACM Conference on Wireless Network Security, 2018
[27] Chakravarty, Sambuddho; Portokalidis, Georgios ; Polychronakis, Michalis ; Keromytis, Angelos D., Detecting traffic snooping in tor using decoys, Workshop on Recent Advances in Intrusion Detection, 2011
[28] Cohen, Fred; Marin, Irwin ; Sappington, Jeanne ; Stewart, Corbin ; Thomas, Eric, Red Teaming Experiments with Deception Technologies, 2001, available at http://all.net/journal/deception/RedTeamingExperiments.pdf
[29] Cohen, Fred; Koike, Deanna, Leading attackers through attack graphs with deceptions., Computers & Security, 2002
[30] Provos, Niels, virtual honeypot framework, USENIX Security Symposium, 2004
[31] Rrushi, Julian L., An exploration of defensive deception in industrial communication networks, Crit. Infrastruct. Protection, 2011
[32] MacFarland, Douglas C.; Shue, Craig A., The SDN Shuffle: Creating a Moving-Target Defense using Host-based Software-Defined Networking, 2015
[33] Han, Wonkyu; Zhao,, Ziming; Doupé, Adam; Ahn, Gail-Joon, HoneyMix: Toward SDN-based Intelligent Honeynet, SDN-NFVSec, 2016
[34] Achleitner, Stefan; La Porta, Thomas F. ; McDaniel, Patrick ; Sugrim, Shridatt ; Krishnamurthy, Srikanth V. ; Chadha, Ritu, Deceiving Network Reconnaissance Using SDN-Based Virtual Topologies, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, 2017
[35] Zhao, Qi, Zhang, Chuanhao, Zhao, Zheng, A decoy chain deployment method based on SDN and NFV against penetration attack, 2017
[36] Urias, Vincent E.; Stout, William M.S. ; Loverro, Caleb ; Anthony, Benjamin, NOW YOU SEE ME, NOW YOU DON'T ADVANCING NETWORK DEFENSE THROUGH NETWORK DECEPTION, 2017
[37] Kyung, Sukwha; Han, Wonkyu ; Tiwari, Naveen ; Dixit, Vaibhav Hemant ; Srinivas, Lakshmi ; Zhao, Ziming ; Doupe, Adam ; Ahn, Gail-Joon, HONEYPROXY: Design and Implementation of Next-Generation Honeynet via SDN, IEEE Conference on Communications and Network Security, 2017
[38] Chen, Mingyong; Wu, Weimin, Research on moving target defense based on SDN, Green Energy and Sustainable Development I, 2017
[39] Chowdhary, Ankur; Alshamrani, Adel ; AZ, Tempe, ; Huang, Dijiang ; Liang, Hongbin, MTD Analysis and evaluation framework in Software Defined Network (MASON), SDN-NFVSec, 2018
[40] Kampanakis, Panos; Perros, Harry ; Beyene, Tsegereda, SDN-based solutions for Moving Target Defense network protection, IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2014
[41] Steinberger, Jessica; Kuhnert, Benjamin ; Dietz, Christian ; Lisa , Ball; Sperotto, Anna ; Baier, Harald ; Pras, Aiko ; Dreo, Gabi, DDoS Defense using MTD and SDN, IEEE/IFIP Network Operations and Management Symposium, 2018
[42] Makanju, Adetokunbo; Zincir-Heywood, A. Nur ; Kiyomoto, Shinsaku, On Evolutionary Computation for Moving Target Defense in Software Defined Networks, GECCO '17 Companion, Berlin, Germany, 2017
[43] Debroy, Saptarshi; Calyam, Prasad ; Nguyen, Minh ; Stage, Allen ; Georgiev, Vladimir, Frequency-Minimal Moving Target Defense using Software-Defined Networking, 2016 International Conference on Computing, Networking and Communications, Cloud Computing and Big Data, 2016
[44] Jafarian, Jafar Haadi; Al-Shaer, Ehab; Duan, Qi, OpenFlow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking, HotSDN, 2012
[45] MacFarland, Douglas C.; Shue, Craig A., The SDN Shuffle: Creating a Moving-Target Defense using Host-based Software-Defined Networking, MTD'15, 2015

[46] Aydeger, Abdullah; Saputro, Nico ; Akkaya, Kemal ; Rahman, Mohammad, Mitigating Crossfire Attacks using SDN-based Moving Target Defense, IEEE 41st Conference on Local Computer Networks, 2016
[47] WolfgangBraun and MichaelMenth, Software-DefinedNetworkingUsingOpenFlow: Protocols, ApplicationsandArchitecturalDesignChoices, 2014
[48] Open Networking Foundation, OpenFlow Switch Specification Version 1.0.0, 2009. Available online: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf.
[49] Open Networking Foundation, OpenFlow Switch Specification Version 1.1.0, 2011. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf.
[50] Open Networking Foundation, OpenFlow Switch Specification Version 1.2, 2011. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf.
[51] Open Networking Foundation, OpenFlow Switch Specification Version 1.3.0, 2012. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.
[52] Open Networking Foundation, OpenFlow Switch Specification Version 1.4.0, 2013. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf.
[53] Open Networking Foundation, OpenFlow Switch Specification Version 1.5.1, 2015. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf